

AIR FORCE



AD-A221 049

HUMAN RESOURCES

**PROCEEDINGS OF THE SECOND INTELLIGENT
TUTORING SYSTEMS RESEARCH FORUM**

Edited by

**Hugh L. Burns, Lt Col, USAF
James Parlett, Major, USAF**

**TRAINING SYSTEMS DIVISION
Brooks Air Force Base, Texas 78235-5601**

Carol Luckhardt

**Southwest Research Institute
6220 Culebra
San Antonio, Texas 78284**

April 1990

Interim Technical Paper for Period June 1988 - June 1989

Approved for public release; distribution is unlimited.

LABORATORY

**AIR FORCE SYSTEMS COMMAND
BROOKS AIR FORCE BASE, TEXAS 78235-5601**

90 04 3007

NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely Government-related procurement, the United States Government incurs no responsibility or any obligation whatsoever. The fact that the Government may have formulated or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication, or otherwise in any manner construed, as licensing the holder, or any other person or corporation; or as conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

The Public Affairs Office has reviewed this paper, and it is releasable to the National Technical Information Service, where it will be available to the general public, including foreign nationals.

This paper has been reviewed and is approved for publication.

HENDRICK W. RUCK, Technical Advisor
Training Systems Division

RODGER D. BALLENTINE, Colonel, USAF
Chief, Training Systems Division

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.				
1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE April 1990	3. REPORT TYPE AND DATES COVERED Interim Report - June 1988 to June 1989		
4. TITLE AND SUBTITLE Proceedings of the Second Intelligent Tutoring Systems Research Forum		5. FUNDING NUMBERS PE - 61101F PR - ILIR TA - 20 WU - 16		
6. AUTHOR(S) Hugh Burns James Parlett Carol Luckhardt				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Training Systems Division Air Force Human Resources Laboratory Brooks Air Force Base, Texas 78235-5601		8. PERFORMING ORGANIZATION REPORT NUMBER AFHRL-TP-89-31		
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSORING/MONITORING AGENCY REPORT NUMBER		
11. SUPPLEMENTARY NOTES				
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.		12b. DISTRIBUTION CODE		
13. ABSTRACT (Maximum 200 words) These proceedings identify, discuss, and (where possible) clarify issues of knowledge representation in the design and implementation of Intelligent Tutoring Systems (ITSs). Various architectures are discussed, including case-based reasoning and hypermedia. In addition, issues of interface design, student modeling, blackboard architectures, and ITS evaluation are dealt with in the presentations provided herein. Specific authors and topics are as follows: Burns and Parlett describe current theories of ITS design and argue for an increasingly integrated and complex approach to training system design; Regian describes an ITS for teaching high-performance tasks using voice synthesis; Fink discusses computational issues and problems associated with knowledge representation in ITSs in high-performance domains; Bonar treats issues in intelligent interface design with particular emphasis on interfaces for intelligent discovery microworlds; Porter et al. focus on explanation generation in ITSs which use very large knowledge bases; Woolf's essay describes the process of encoding tutoring; knowledge in a knowledge-based tutor and breaks that process down into its component parts; O'Neil et al. present a design model for domain-independent instructional strategies in ITSs; Swigger presents issues surrounding the design, implementation, and evaluation of flexible interfaces for training environments; Riesbeck et al. argue for a new focus on Creative Tutoring Systems which use case-based reasoning and representational architectures in their design; Pirolli discusses a				
14. SUBJECT TERMS Artificial Intelligence Computer-Based Training Intelligent Tutoring Systems		15. NUMBER OF PAGES 170		
		16. PRICE CODE		
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL	

Item 13. (Concluded)

hypermedia-based approach to the complex problems of instructional design; Hull discusses literacy skills (both traditional and computer-based) which impact on the potential effectiveness of computer-based tutoring; and Baker explores the roles of technology assessment with respect to training environments.

**PROCEEDINGS OF THE SECOND INTELLIGENT
TUTORING SYSTEMS RESEARCH FORUM**

**Edited by
Hugh L. Burns, Lt Col, USAF
James Parlett, Major, USAF**

**TRAINING SYSTEMS DIVISION
Brooks Air Force Base, Texas 78235-5601**

**Carol Luckhardt
Southwest Research Institute
6220 Culebra
San Antonio, Texas 78284**

Reviewed and submitted for publication by

**Hugh L. Burns
Chief, Intelligent Systems Branch**

This publication is primarily a working paper. It is published solely to document work performed.

SUMMARY

This paper provides the entire proceedings of the Second Intelligent Tutoring Systems Research Forum, which was sponsored by the Air Force Human Resources Laboratory with the assistance of Southwest Research Institute, and which was held in San Antonio on 6-7 April 1989. The purpose of the Forum was to assemble 11 invited speakers--all experts in the field of artificial intelligence in training--to speak to over 100 representatives of the Department of Defense, academia, and industry on the problems of representing various kinds of knowledge within an Intelligent Tutoring System (ITS). Among these kinds of knowledge are domain expertise (the knowledge to be taught to the student), instructional knowledge (knowledge of how to teach), and the student model (knowledge about who is being taught). Representation of each kind of knowledge presents special computational challenges to a system designer; further, integrating these knowledge types so that the tutoring system can begin to approximate the high quality of good human instruction also presents significant challenges.

Related peripheral issues are also addressed in these proceedings. Some of these issues include measuring and evaluating the effectiveness of ITSs, and defining the changing roles of literacy and text in intelligent computer-aided instruction.

The central thrust of this volume is toward recognition of ITS design as an increasingly complex and integrated task involving multidisciplinary teams of experts in domain, instruction, and assessment. The early days of ITS design provided us with a cohesive component-based architecture for the design of ITSs; in these papers, however, the distinctions between such components blur and break down as designers attempt to capture a widening range of the complex cognitive tasks involved in instruction.

Accession For	
NTIS CR&I	<input checked="" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution /	
Availability Codes	
Dist	Avail and/or Special
A-1	



PREFACE

The purpose of these Proceedings is to document, explore, and clarify the dimensions of knowledge representation in the development of Intelligent Tutoring Systems (ITSs). Knowledge representation is a critical issue in the design of ITSs; this Forum presents multiple viewpoints on this issue. In doing so, the Forum itself (and these Proceedings) represents a set of stimuli to existing research and development efforts--both contractual and in-house--for the Training Systems Division, and well as to other Department of Defense research organizations (such as the Army Research Institute and the Office of Naval Research). We expect that these Proceedings will provide new directions for thought and investigation for several years to come.

We gratefully acknowledge the contributions of all the speakers represented in these Proceedings. In addition, we acknowledge the administrative support of Ms. Rose Reyes, and the scientific and technical support of Mr. Frank Hughes and Drs. Joseph Psotka, Susan Chipman, and Kurt Steuck, who chaired sessions of the Forum. Finally, we are grateful for the support of the many employees of Southwest Research Institute who contributed to the success of the Forum in various significant but hidden ways.

TABLE OF CONTENTS

	Page
THE DIMENSIONS OF TOMORROW'S TRAINING VISION: ON KNOWLEDGE ARCHITECTURES FOR INTELLIGENT TUTORING SYSTEMS	1
REPRESENTING AND TEACHING HIGH-PERFORMANCE TASKS WITHIN INTELLIGENT TUTORING SYSTEMS	11
ISSUES IN REPRESENTING KNOWLEDGE FOR TRAINING HIGH PERFORMANCE SKILLS	23
INTERFACE ARCHITECTURES FOR INTELLIGENT TUTORING SYSTEMS	39
GENERATING EXPLANATIONS IN AN INTELLIGENT TUTOR DESIGNED TO TEACH FUNDAMENTAL KNOWLEDGE	55
REPRESENTING, ACQUIRING, AND REASONING ABOUT TUTORING KNOWLEDGE	71
DESIGN OF A DOMAIN-INDEPENDENT PROBLEM SOLVING INSTRUCTIONAL STRATEGY FOR INTELLIGENT COMPUTER-ASSISTED INSTRUCTION	91
MANAGING COMMUNICATION KNOWLEDGE	103
FROM TRAINING TO TEACHING: TECHNIQUES FOR CASE-BASED ITS	117
ON THE ART OF BUILDING: PUTTING A NEW INSTRUCTIONAL DESIGN INTO PRACTICE	131
LITERACY AS PREREQUISITE KNOWLEDGE	145
TECHNOLOGY ASSESSMENT: POLICY AND METHODOLOGICAL ISSUES	153

THE DIMENSIONS OF TOMORROW'S TRAINING VISION: ON KNOWLEDGE ARCHITECTURES FOR INTELLIGENT TUTORING SYSTEMS

Hugh Burns, Lt Col, USAF
Chief, Intelligent Systems Branch

James Parlett, Major, USAF
Deputy Chief, Intelligent Systems Branch

Air Force Human Resources Laboratory
Brooks AFB, TX 78235

In the twenty-first century, a technical trainer's professional credibility will depend in part on how well he or she has kept up with intelligent training systems. Plainly stated, technical trainers cannot afford artificial intelligence illiteracy in tomorrow's electronic schoolhouses. They cannot afford it for their professional lives; they certainly cannot afford it for their students' futures. An "intelligent" computer of some sort is clearly on the practical horizon. Technical trainers need to be able to exploit it, and--better yet--need to be afforded the opportunity to help design, develop, test, and evaluate intelligent training systems. But until training instructors and managers have some empirically validated proof of the new "intelligent" training vision research scientists imagine, intelligent tutoring systems (ITSs) will not be wisely and widely used. Researchers and users must therefore come together frequently in forums to learn, to clarify, to explore, and to discuss the dimensions of tomorrow's training vision.

Designing, developing, and evaluating intelligent tutoring systems is not so much a strategic matter--what things to do--as a technical enterprise--how to do things. Figure 1 portrays the overall evolving architecture of a practical intelligent tutoring system.

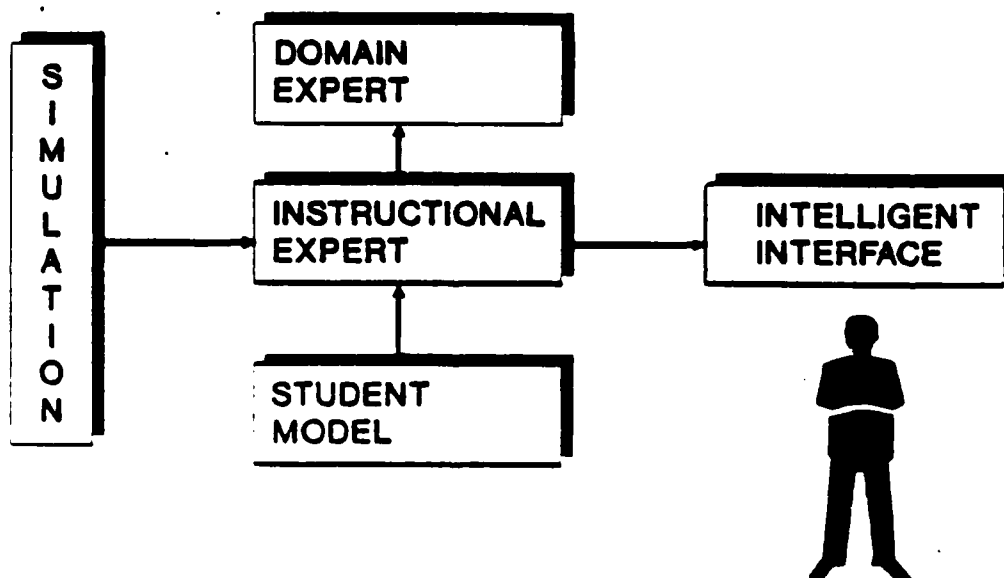


Figure 1. Intelligent Tutoring System Architecture.

The research and literature have laid out a fundamental, highly interactive set of components. In *Foundations of Intelligent Tutoring Systems* [8], the foundational anatomy was used to discuss research issues within each of the separate components--though all of the authors knew how difficult it was to separate, if you will, the dancers from the dance. In the forum this year, the issues have been broadened to emphasize the interactivity of the major components in the design. This year we will be exploring the dimensions of expertise, instruction, literacy, and application.

Now, here is what we need a better understanding of:

- * **Domain Knowledge: The Expert Dimensions**
- * **Teaching Knowledge: The Instruction Dimensions**
- * **Communication Knowledge: The Literacy Dimensions**
- * **Meeting Users' Needs: The Application Dimensions**

ON DOMAIN KNOWLEDGE: THE EXPERT DIMENSIONS

Trends in technical training today suggest a balanced view of declarative knowledges and procedural skills. Instructors are articulate about the domain facts, as well as the skills they wish to see demonstrated. Many are also concerned about the learning processes students use to master various technical skills, such as the specific troubleshooting or operational procedures the trainees use to solve the problem. As such trends continue, then intelligent tutoring systems can be valuable allies.

The current design of ITS is closely tied to how an ITS presents the domain to a student. Figure 2 highlights the dimensions of device or operational simulation, the domain expert, and the intelligent interface.

Domain knowledge is most often represented operationally. Most ITS design is featured on the user interface. The domain obviously has implications for how a tutor is conceptualized and how a student views the instruction. Most importantly, the knowledge acquisition investments for domain representation are the major roadblocks in tutorial development.

The design of the domain knowledge structures includes depicting properties of the domain itself and the tasks to be trained. Some domain knowledge is unstructured, but most domain knowledge is highly structured and specific. Theoretical knowledge might appear in a large, multi-user knowledge base in a more structured fashion; for example, presenting a particular viewpoint within the knowledge base that has been created based on the needs of a specific goal for using the knowledge base. Some tasks have no generic viewpoint; equipment design, for example, may implicitly represent the important features in an operations console. Even in domains that do not have a task-defined interface--such as programming, algebra, electronic troubleshooting, or writing--relevant and often creative representations can be designed and presented. Representations are based on how experts understand and interpret the task domain, and potential areas of development will demonstrate static, procedural, and even "automatized" tutors.

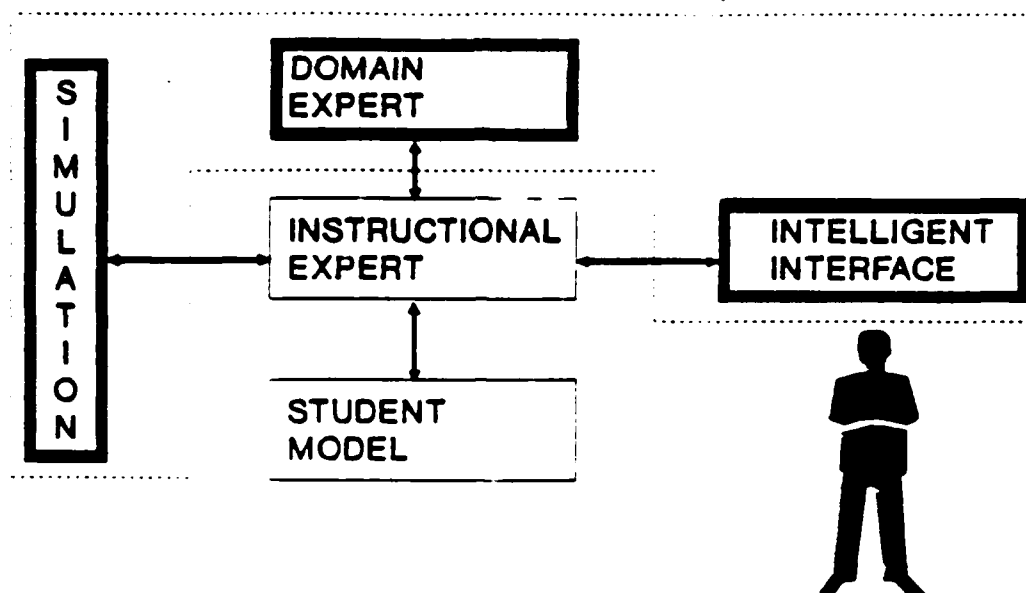


Figure 2. Domain Knowledge Architecture.

How domain knowledge is represented and used in a computer program depends on the task to be performed. The task as it breaks down into goals of a training system provides a viewpoint about the basic knowledge in the domain. For example, in the domain of jet engine mechanics, the task of diagnosing malfunctions generates various viewpoints of the knowledge about aircraft, mechanics, and electronics. Such viewpoints might include a functional representation that fosters a mental simulation of how the engine works or an experientially based representation which provides quick condition-action, pattern matching capabilities. Some high-performance domains require viewpoints on the knowledge that include physical skills.

An intelligent tutoring system can help a technical instructor train more efficiently, but a huge question remains: Can an intelligent tutoring system help trainees learn how to perform their real jobs more effectively? The honest, intuitive answer is: "It depends." A more honest, intellectual answer is, "We do not have enough empirical data to cite a significant difference." However, if both students and instructors form an alliance to use intelligent training systems appropriately, then students should perform their jobs more effectively. Some preliminary evidence suggests that technical trainers often have not developed an adequate sense of how specific technical specialties can be viewed in this larger job/task context, and so have not been able to pass that "big picture" knowledge on to students.

Intelligent computer-assisted instruction is often controversial because it takes an explicit domain point of view. Its value is determined by how robustly the instructional environment captures the domain design, how "smartly" the domain expert is represented, and how capably and flexibly the instructional intuitions are represented within the machine. This evolution, however, means that in the near future intelligent training systems are more likely to be appreciated for the "tools" which are emerging for domain design and delivery rather than as "tutors" in and of themselves.

ON TEACHING KNOWLEDGE: THE INSTRUCTIONAL DIMENSIONS

The single, major advantage of an intelligent tutoring system is simply achieving a more favorable teacher-student ratio. One-on-one tutoring is not new. Any teacher or coach who is trying to improve performance of a skill can attest that the more time they can spend one-on-one, the greater the likelihood that a student's performance will improve. So, intelligent tutoring systems potentially allow more one-on-one instructional efficiency and, thereby, leverage more instructional effectiveness. But there are challenges.

Proposed architectures for representing teaching knowledge in intelligent tutoring systems can be described both in terms of how that knowledge can be understood and how it can be represented as sets of domain-independent tutoring strategies. Teaching knowledge can also be used to develop methods to generate answers and explanations from instructional knowledge bases in which a coherent viewpoint is tailored to the individual student's needs. From a researcher's point of view, explanation generation with and without the benefit of a student model could empirically evaluate the robustness of the student model itself. Consequently, tools to encode an instructional expert to control knowledge in intelligent tutors will involve representing multiple knowledge concepts and proposing alternative teaching tactics. Figure 3 depicts the interaction between the instructional expert module and a dynamic student model, as well as the delivery implications for a learner-friendly intelligent interface.

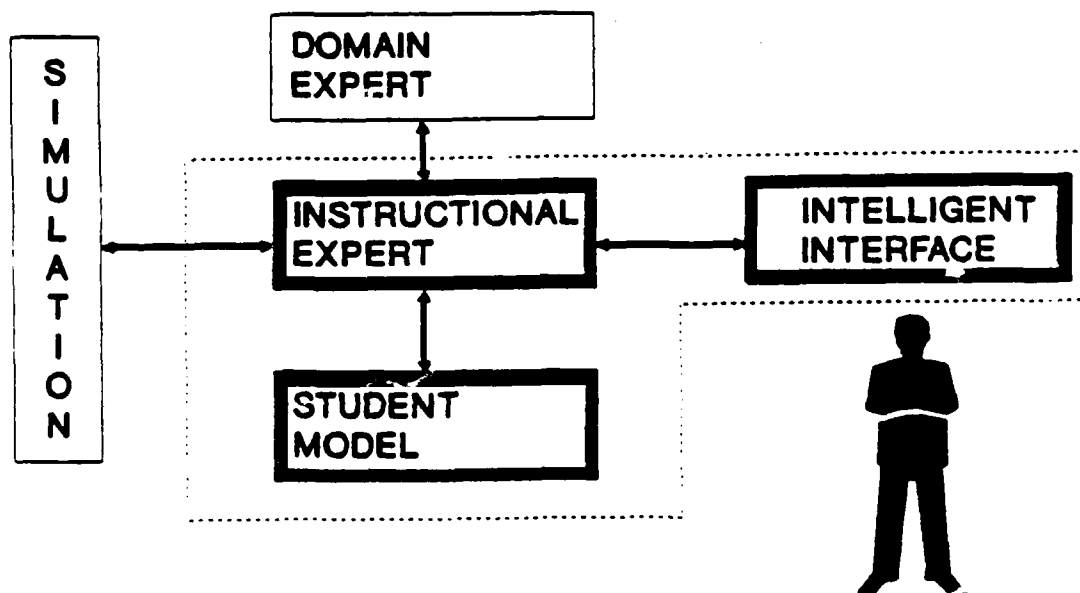


Figure 3. Teaching Knowledge Architecture.

Many technical trainers are convinced that today's inflexible or brittle software does not significantly help them meet the needs of their students. Not many technical trainers are convinced that the computers reach individual students with individual help. Much of this brittle software sits on the shelf in the "training research laboratory" waiting to be prescribed. The

solution is to create flexible software that permits teacher-controlled modifications. An intelligent tutoring interface should be an electronic mirror reflecting what students want to do and what instructors want students to do--allowing both to think more about their choices. In whatever way a technical training instructor perceives skill development, an intelligent training system must allow appropriate opportunities for instructors to intervene precisely in the student's learning process. Instructors also want more control over supplementary materials; they want to have authority over software prescriptions. Because they want the capability to reinforce their students personally, instructors should have the capability to customize software and to at least influence the architecture of an instructional expert module.

In the near term, computers will provide recordings of a student's problem-solving processes so that students may observe their own performances, as well as the performance of experts. Such ITS experiences will offer a time-compressed view of the tactics involved in solving a technical problem. The "compression of process" will allow for awareness of past habits to more fully inform future problem-solving. Imagine seeing electronic troubleshooting expertise in motion. In the long term, as more and more "artificial intelligence" is designed into the instructional expert, applications will be even more individualized. These trends are unmistakable. Researchers must keep investigating how humans learn and teach so that intelligent computers will surprise us even more in the future.

The tutors which educators and technical trainers will use must give students a strategic sense of purpose, must encourage them to ~~work appropriately~~ with an outcome in mind, must coach them to perform efficiently, and must have them recognize effective job performance.

ON COMMUNICATION KNOWLEDGE: THE LITERACY DIMENSIONS

Intelligent tutoring systems that are capable of training students in complex problem-solving tasks require human/machine interfaces that are extremely flexible. "Interactivity" is undoubtedly the real strength of and hope for intelligent tutoring systems. The set of instructional activities in an intelligent tutoring system provides a way for investigating, exploring, and stimulating the communication processes in learning.

In many human/machine systems, the parameters and requirements of the communication protocol are well understood. Databases, for example, "expect" users to ask questions which the system is programmed to answer. Traditional computer-based instruction asks questions, and students answer. But our vision of communication between a human and an intelligent tutoring system is far more demanding. An intelligent tutoring system should be able to tell stories and run simulations. It must be able to explain itself in an accessible way. It must be able to communicate its own "expectations" to a student. It must "listen." In short, an intelligent tutoring system must be able to perform many of the communication skills handled so fluently by human teachers. But how? The design demands of intelligent tutoring systems--with so many graphics options, so many text and language choices--make it almost impossible for domain experts to select a single communication model.

Figure 4 portrays how the communication architecture expresses the necessary relationship between the user and the interface, all under the control of the instructional expert.

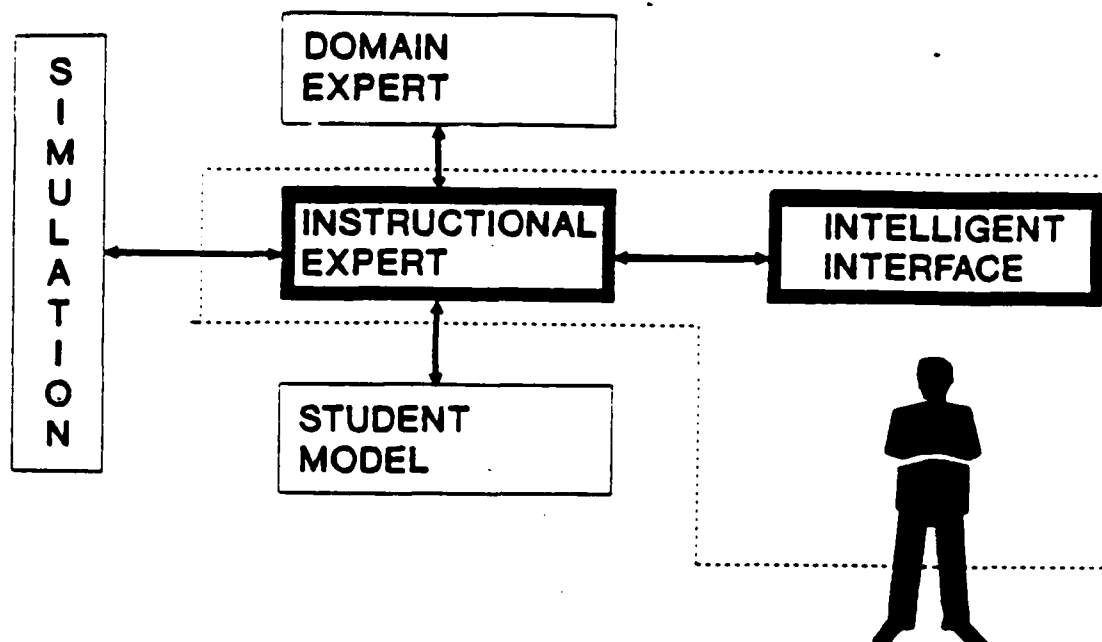


Figure 4. Communication Knowledge Architecture.

Yes, interactivity should be incorporated into ITS dialogues, but what techniques should be used to achieve them? Language-based? Graphics-based? Both? Should students type their own short answers in their own words rather than selecting them from a menu of multiple choices? Should students type explanations for their answers? What about knowledge-based interactions? Should students be able to try many alternatives? Should students be able to create and play and explore? Some student/computer interactions are moving toward more student-centered, reactive learning environments. Microworlds demonstrate the effectiveness of such instructional environments. Clearly, any ITS will need a library of examples, including examples of failures and exceptions. And such an ITS will necessarily be versatile in its presentation of those examples, providing the student with multiple occasions in which his or her creativity, problem-solving, and learning skills might be developed.

In the absence of such occasions for creativity, exploration, and self-generated questions, the student will have no mental "hooks" on which to hang his or her burgeoning knowledge.

Such creative tutoring systems will require that students ask and systems answer, but in such a way that the student must ask more and fruitful questions, that students explore "garden paths," that students enter willingly and fully into the problem-solving sessions. In other words, such tutoring systems must ultimately communicate a dramatic sense of play, exploration, and instruction to a student. To engage students in learning, then, tutoring systems must enact and communicate a clear and versatile pedagogical stance.

ON MEETING USERS' NEEDS: THE APPLICATIONS DIMENSIONS

To effectively meet the varied needs of ITS users, we must first consider the identity of potential ITS users. The most important ITS users—learners—must have prerequisite skills,

knowledges, and epistemologies in order to learn from an ITS. **Instructors** must also have ways of managing and modelling curricula involving ITS in a way that promotes efficient, effective, and flexible instruction. Finally, **decision makers** must have credible information about the effectiveness of intelligent instructional technology in training and educational settings.

Figure 5 portrays the concept of multiple intelligent tutoring systems meeting these users' needs in one-on-one instructional settings. Here the emphasis is on the flexibility of the intelligent systems to deliver cost-effective and instructionally effective training. And from the interactions of the learners with an ITS, instructors and decision makers (and ultimately, the ITS itself) will themselves learn to better meet the learner's needs.

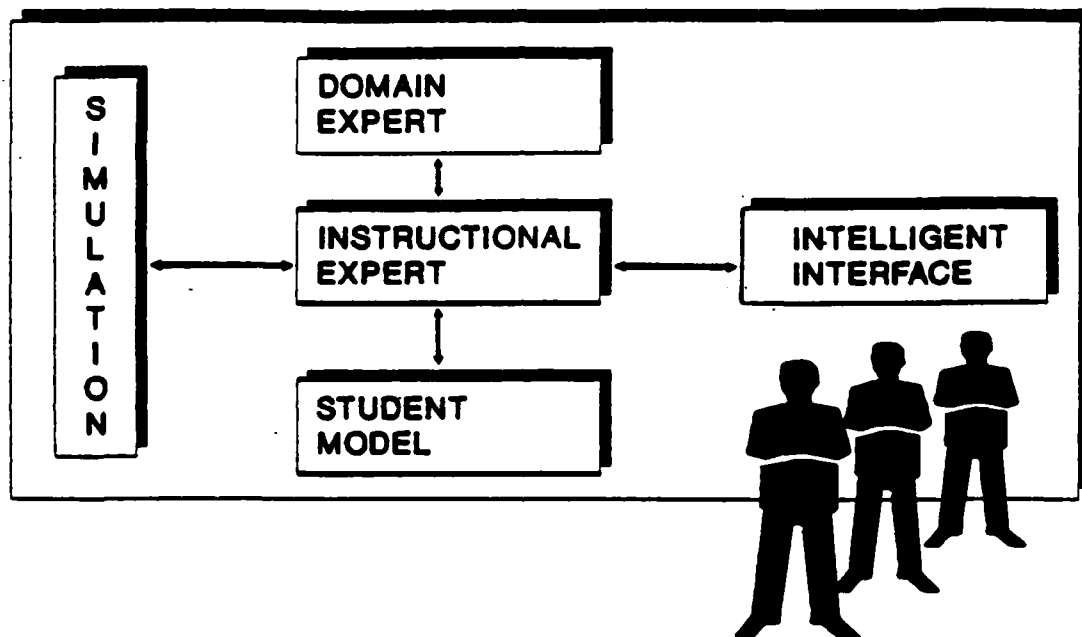


Figure 5. Meeting Users' Needs: Flexibility and Effectiveness.

Current literacy theory—ideas about how people learn to read and write—must inform the construction and evaluation of intelligent tutoring systems. The multiple perspectives of students, many of whom are underprepared in literacy skills (not to mention computer skills), must be accounted for. Understanding literacy today means understanding a plurality of literacies. Good reading or good writing varies across cultural and disciplinary communities and, undoubtedly, the relationship between literacy and educational technology will increase as new technologies such as hypertext and hypermedia systems are introduced into the construction and design of intelligent tutoring systems. Plainly stated, we cannot assume that students possess requisite reading and writing skills for complex problem-solving tasks, such as operating and learning from ITSs. Neither can we assume that the literacy of one group or culture maps clearly onto the literacy implicit (or explicit) in the design and operation of technology in general, and ITS in particular.

To realize the promise of intelligent tutoring systems, teachers must be confident in their own abilities. Further, teachers and trainers must develop new, prescriptive models of instructional design and delivery. Instructional sciences and arts will expand as this novel technology and theories emerge. Instructional design itself will be conceived of as a problem-solving framework. Within this framework, ITS will provide a means to discuss the processes and knowledges involved in training and education.

How will ITS technology transfer from promising demonstrations to real training settings? Modern-minded instructors may eventually wish to advise decision makers about technical curriculum changes, domains, device simulations, subject-matter experts, instructional routines, and interventions, as well as evaluation and security matters. Teachers, learners, and planners alike must constantly engage in a process of technology assessment, a process of examination that leads to a clear and effective transfer of ITS technology from the research laboratory to the classroom.

This technology transfer must involve validating methodologies, articulating appropriate variables, gauging the social settings, as well as considering the range of external motivations--to include the political uses of such assessments. Another related technology transfer complication is cost, both in terms of time to become well acquainted with the various intelligent software packages and in terms of the funds required to purchase and maintain artificial intelligence software. Decision makers need to be given time to learn about specific operating systems, to survey the diverse tutoring applications, to read the documentation, and to plan potential intelligent tutoring applications. Like teachers, decision makers must know what it means to learn. In addition, instructors need training and training support--just like anyone else from whom expertise is expected. Very few military, industry, and even academic managers are promoting adequate training for their faculties and staffs in the area of advanced training technologies; they assume instead that faculty will train themselves if they care enough. Intelligent training systems help students, instructors, and decision makers only if wise investments of time, money, and personnel are made at all levels.

THE DIMENSIONS OF TOMORROW'S TRAINING VISION: A FINAL PERSPECTIVE

Can an intelligent tutoring system help a learner become a better technician, a skilled job performer more in touch with the processes of equipment diagnosis, equipment operation, and organization support? Yes, and technical trainers must lead novices to this journeymen's realization. The issue is not just whether students have improved individual performances but, more importantly, whether or not they have developed and internalized more effective ways of learning while in the act of performing their technical jobs.

The outlook for intelligent tutoring systems is bright, though technical trainers face quite a charge. In the next few years, however, such responsibilities will seem more and more natural.

If technical training instructors have complex domains to simplify, then they can use expert representations and advanced integrated interfaces masterfully. If instructors better understand the nature of teaching, then they can design machines which can tutor effectively. If instructors

better understand the dynamic features of **communication**, then they can individualize instruction efficiently. If trainers and researchers alike are committed to **meeting users' needs**, then they can provide the right people for the right jobs at the right time.

Intelligent tutoring systems--tools for the twenty-first century--are machines which improve and evolve. With them or without them, the Air Force will fly, the Navy will sail, the Marines will land, and the Army will roll. But with wise decisions about how intelligent training systems will supplement technical training, operational training, team training, embedded training, and exportable training, the very enterprise of learning complex technical skills promises to be more efficient, more effective, and much more exciting.

BIBLIOGRAPHY

1. Andriole, S.J., & Hopple, G.W. (Eds.). (1988). *Defense applications of artificial intelligence: Progress and prospects*. Lexington, MA: D.C. Heath.
2. Feigenbaum, E., McCorduck, P., & Nie, F. (1988). *The rise of the expert company*. New York: Time Books.
3. Gardner, H. (1985). *The mind's new science: A history of the cognitive revolution*. New York: Basic Books.
4. Kearsley, G.P. (Ed.). (1987). *Artificial intelligence and instruction: Applications and methods*. Reading, MA: Addison-Wesley.
5. Lawler, R., & Yazdani, M. (Eds.). (1987). *AI and education*. Norwood, NJ: Ablex.
6. Psotka, J., Massey, D., & Mutter, S. (Eds.). (1988). *Intelligent tutoring systems: Lessons learned*. Hillsdale, NJ: Lawrence Erlbaum.
7. Richardson, J.J. (Ed.). (1985). *Expert systems in maintenance*. Park Ridge, NJ: Noyes.
8. Richardson, J.J., & Polson, M. (Eds.). (1988). *Foundations of intelligent tutoring systems*. Hillsdale, NJ: Lawrence Erlbaum.
9. Schank, R.C., & P.G. Childers. (1984). *The cognitive computer: On language, learning, and artificial intelligence*. Reading, MA: Addison-Wesley.
10. Sleeman, D., & J.S. Brown (Eds.). (1982). *Intelligent tutoring systems*. New York: Academic Press.
11. Wenger, E. (1987). *Artificial intelligence and tutoring systems: Computational and cognitive approaches to the communication of knowledge*. Los Altos, CA: Morgan Kaufman.
12. Winograd, T., & Flores, F. (1986). *Understanding computers and cognition: A new foundation for design*. Norwood, NJ: Ablex.
13. Winston, P.H., & Prendergast, K.A. (Eds.). (1984). *The AI business: Commercial uses of artificial intelligence*. Cambridge, MA: MIT.

REPRESENTING AND TEACHING HIGH-PERFORMANCE TASKS WITHIN INTELLIGENT TUTORING SYSTEMS

J. Wesley Regian
Senior Scientist
Intelligent Systems Branch
Air Force Human Resources Laboratory

ABSTRACT

Intelligent Tutoring Systems (ITSs) are advanced Computer-Based Training (CBT) systems which utilize Artificial Intelligence (AI) technology to allow highly individualized instructional interactions with students (Soloway & Littman, 1986; Wenger, 1987; Yazdani, 1986). This paper describes a currently implemented Intelligent Tutoring System (ITS) developed at the Air Force Human Resources Laboratory (AFHRL) that teaches cognitive skills associated with performing an instrument-only landing in a fighter airplane. The ITS was developed not as an actual training device for instrument flight but as a testbed for the application of AI to training in a class of task domains (and task components) that have been referred to as high-performance tasks (Regian & Shute, 1988) and real-time tasks (Ritter & Feurzeig, 1988). In high-performance tasks, there is more of a requirement for speeded, reliable, and automatic task performance than is found in the typical knowledge-rich ITS domains (e.g., medical diagnosis, electronic troubleshooting). The Instrument Flight Trainer (INFLITE) trains students to land a simulated aircraft (F-16) using instruments only. During the process, an intelligent coach monitors the student and provides guidance just as an instructor pilot might guide a student pilot. This guidance is presented verbally, using a speech synthesis device to simulate human speech. The system supports a variety of instructional approaches including the capability to freeze the simulation to give guidance, prebrief students before training sessions, generate guidance in real time during training sessions, debrief students after training sessions, anticipate student errors in real time based on prior student performance, and generate part-task drills to achieve performance goals.

INTRODUCTION

Researchers at AFHRL are applying a taxonomy of learning skills (Kyllonen & Shute, 1987) to the pedagogical issues surrounding ITS design and development. The taxonomy provides a means of categorizing target domains and consequently specifying the appropriate training approaches for particular ITSs. This paper focuses on an area that has only recently been investigated by the ITS community: a class of tasks referred to as high-performance tasks (Regian & Shute, 1988). In high-performance tasks, there is more of a requirement for speeded, reliable, and automatic task performance than is found in the typical knowledge-rich ITS domains (e.g., medical diagnosis, electronic troubleshooting). This paper describes an INFLITE developed in the Training Systems Division at AFHRL to evaluate methods for using AI to train high-performance tasks.

INFLITE is in no sense a serious attempt to develop an application-ready instrument flight training device. Rather, INFLITE is a testbed system designed for the purpose of evaluating

promising approaches to training high-performance tasks. The decision to use flight simulation as the prototype domain was guided by a desire to use an inherently interesting task to increase motivation in experimental subjects.

INTELLIGENT TUTORING SYSTEMS

Computer-Aided Instruction (CAI) is a mature technology used to teach students in a wide variety of domains. The introduction of AI technology to the field of CAI has prompted research and development efforts in an area known as Intelligent Computer-Aided Instruction (ICAI). In some cases, ICAI has been touted as a revolutionary alternative to traditional CAI. "With the advent of powerful, inexpensive school computers, ICAI is emerging as a potential rival to CAI" (Dede & Swigger, 1987). In contrast to this, one may conceive of CBT systems as lying along a continuum which runs from CAI to ICAI. Thus, ICAI may be seen as an evolution of CAI rather than as a revolutionary alternative. The key difference between the two perspectives is that in a revolution the old guard is dismissed and replaced, whereas in an evolution the old guard is a foundation to be built upon. An important implication of the "evolution" perspective is that we are less likely to throw out the strengths and accomplishments of the old guard. This perspective does not imply, however, that there are no important differences between CAI systems and ICAI systems.

For my purposes, I discriminate among CBT systems according to the degree to which the instruction they provide is individualized. My choice of this particular dimension is based on more of a desire for utilitarianism than for precision. A great deal of data from the educational literature indicates that carefully individualized instruction is superior to conventional group instruction (Bloom, 1984; Woolf, 1987). Thus, an important way in which CBT systems differ is in the degree to which their behavior is modified by an inferred "model of the student's current understanding of the subject matter" (VanLehn, 1986). The CBT system that is less intelligent by this definition, I conceive of as CAI. Similarly, the system that is more intelligent, I conceive of as ICAI. Often, ICAI systems are referred to as "Intelligent Tutoring Systems, or ITSs" (Sleeman & Brown, 1982). This term is particularly appropriate, as it brings to mind one-on-one tutoring.

With respect to individualization, it is important to note that virtually all traditional CAI systems are individualized in the sense that they are self-paced, and many are further individualized by virtue of branching routines which allow different students to receive different instruction. CAI systems with branching routines are, in fact, more individualized than those without branching routines. Thus, they are more intelligent by the current definition (although in a weak sense, as we shall see). Nevertheless, in branched CAI the instructional developer must explicitly encode the actions generated by all possible branches, and there is a finite number of possible paths through these branches. As one moves further away from the CAI to the ICAI end of the continuum, one begins to see a very different and more powerful approach to individualization. This more powerful approach is touched on by Wenger (1987) when he refers to explicit encoding of knowledge rather than encoding of decisions (p. 4). An ITS (a term which probably should be reserved for systems which are very far toward the ICAI end of the continuum) utilizes a diverse set of knowledge bases and inference routines to "compose instructional

interactions dynamically, making decisions by reference to the knowledge with which they have been provided" (Wenger, 1987, p. 5).

The ITS Anatomy

In an ITS, individualized instruction is an emergent property of several interacting components. ITSs often consist of four, sometimes five, distinct components. These are the expert module, the instructional module, the student model, the interface, and often a device simulation or other instructional environment.

The expert module is a programmed representation of expert knowledge in the target domain (that which is being taught). It is almost identical to what is commonly known as an expert system, except in this context it is often very articulate (able to generate some form of rationale for its actions) and capable of generating alternative solution paths (rather than a single "best" path). The expert module brings domain knowledge to the ITS. In some useful sense, the system "knows" how to perform the task which it is seeking to teach, and can demonstrate that knowledge.

The instructional module is a programmed representation of expert knowledge on pedagogy in the target domain. It is generally not articulate but is usually capable of generating alternative instructional approaches based on the current knowledge level of the current student. Although the expert module typically derives from knowledge engineering accomplished with an expert practitioner in the target domain, the instructional module may derive from knowledge engineering accomplished with an expert instructor in the target domain (which may or may not be the same person as the expert practitioner), with a general training specialist, or both.

The student model constitutes a repository for information about each student that uses the system. It differs from the expert and instructional modules in that it is a mere shell at the beginning of an initial tutoring session, whereas the latter two are generally complete when the development of the ITS is complete. At the beginning of an initial tutoring session, the student model is merely a place to store specific kinds of information about students in particular formats that will be useful for the instructional module to access. The student model is dynamically updated during tutoring sessions to maintain current information about the student such as what the student knows, what the student does not know, and misconceptions the student may have. The student model brings situational awareness to the ITS. Thus, the system "knows" whom it is teaching and can make informed decisions about what to teach next and how to teach it.

The interface provides the methods by which the student interacts with the ITS. The interface may include such output methods as computer-generated graphics and text, recorded video images, or speech synthesizers; and such input devices as a mouse, keyboard, touchscreen, joystick, or voice recognition system. One important point about the interface is that it should be as simple as possible so that learning to use the ITS does not interfere with learning from the ITS.

Many ITSs (e.g., STEAMER, IMTS/Bladefold, Sherlock) use an embedded computer simulation of an electrical or mechanical device to provide an instructional context, or environment. That is, the device simulations are used to teach operation or maintenance of a specific device in the context of an operating model of the device. Other ITSs teach a body of knowledge that is not specific to any particular device, and yet they use other kinds of simulations to provide instructional environments. For example, Smithtown uses a simulation of microeconomics operating in a small town, and the Orbital Mechanics tutor uses a simulation of orbital dynamics.

Knowledge-Rich Versus High-Performance Domains

Traditionally (if the term applies to a technology less than 20 years old), ITSs have focused on knowledge-rich domains such as electronic troubleshooting, physics, economics, and medical diagnosis. Furthermore, they have focused on the higher-level problem-solving components of these domains even though knowledge-rich domains almost invariably involve components of expertise, sometimes called enabling skills, which can be characterized as high-performance components. For example, electronic troubleshooting involves schematic-tracing, which is supported by the ability to immediately and accurately combine gate inputs to determine the output of a particular gate type as represented on the schematic. Similarly, expert performance in theoretical physics requires total facility with basic math and algebraic skills. Human instructors can recognize deficiencies in basic enabling skills (especially in one-on-one tutoring situations) and apply methods to correct these deficiencies.

ITSs as a rule are not sensitive to deficiencies in basic enabling skills, even though they are not difficult to identify. Moreover, computers are particularly well suited to providing the kind of drill-and-practice exercises that can correct the deficiencies. In generating instruction for knowledge-rich domains, ITSs may be sensitive to the full range of performance determinants for the domain, and have appropriate routines available for remediation. Furthermore, there is a place for ITS technology even in primarily high-performance domains such as air traffic control, air intercept control, typing, mission control console operation, and simple equipment operation.

TUTORS FOR HIGH-PERFORMANCE TASKS

Any training program should be designed with an awareness of the underlying cognitive operations that support performance in the targeted task or domain. Tasks may depend on greater or lesser contributions from declarative knowledge, procedural knowledge/skill, or performance skill determinants. These categories of cognitive operations may be said to lie along a continuum which runs from more knowledge-based to more performance-based (see, for example, Kyllonen & Shute, 1987). Though most complex tasks are supported to some degree by all of these categories of operations, many tasks are heavily weighted toward one end of the continuum. Some tasks, for example, are very knowledge-based, such as electronic troubleshooting or medical diagnosis. Other tasks tend to be much more performance-based, such as air intercept control or typing. Although both types of tasks require a certain amount of knowledge to support performance, knowledge-rich tasks require greater depth and breadth of knowledge and are less reliant on performance-based skills. The more performance-based tasks, on the other hand, often require key task components to be cognitively automatized to

the point where task performance is smooth, fluid, and effortless. Such an assimilation, or "automatization," of the task has important benefits. For example, automatized task performance allows the individual to perform other functions at the same time and renders task performance highly reliable under stress (Schneider & Shiffrin, 1977) and highly resistant to skill degradation (Regian & Schneider, 1986).

The Role of Automaticity in High-Performance Skill Training

The automatic/controlled processing framework (Shiffrin & Schneider, 1977) provides a theoretical approach to training high-performance skills. The framework posits two qualitatively different forms of processing that underlie human performance. Automatic processing is fast, parallel, fairly effortless, not limited by short-term memory capacity, not under direct subject control, and used in performing well-developed skilled behaviors. This mode of processing develops when subjects deal with training stimuli in a consistent manner over many trials. Controlled processing is slow, effortful, capacity-limited, subject-controlled, and used to deal with novel, inconsistent, or poorly learned information. This mode of processing is expected at the beginning of practice on any novel task, and throughout practice when a subject's response to a stimulus varies from trial to trial. In this framework, high-performance skills are trainable because they involve components that can be executed rapidly, reliably, and with little effort, freeing cognitive resources for performing other non-automatic tasks (see Schneider, 1985).

In designing training procedures for high-performance tasks, two important findings from the automatic/controlled framework should be considered. The first centers on the distinction between consistent practice and varied (or inconsistent) practice. Consistent practice produces substantial improvements in performance as automatic processing develops (e.g., 98% reduction in visual search comparison rates, Fisk & Schneider, 1983). Varied practice utilizes only controlled processing and produces little improvement in performance (e.g., no change in letter search performance over 4 months of training, Shiffrin & Schneider, 1977). The second finding centers on the amount of effort required to perform automatic processing tasks. Consistent practice greatly reduces the amount of effort required to perform a task, allowing controlled processing to be allocated to another task. When subjects have already developed automatic processes to perform one task, they can learn to time-share another task with little or no deficit. After 20 hours of consistent practice in two search tasks, subjects were able to perform both tasks simultaneously nearly as well as they could perform each separately (Fisk & Schneider, 1983; Schneider & Fisk, 1982a, 1982b, 1984). In addition, automatic task performance has the advantage of being far more reliable under stress (see Hancock & Pierce, 1984).

The acquisition of skill with practice is assumed to result from the development of automatic processes which are used to perform consistent task components. Any applied skill of reasonable complexity is likely to involve both consistent and inconsistent components. Thus, an empirically verified componential breakdown of a complex skill is useful for training. Performance on consistent components is likely to change significantly over extensive practice, whereas performance on inconsistent components is likely to asymptote relatively quickly. Automatization of consistent components has the benefit of freeing up processing capacity that may then be

applied to inconsistent components. Furthermore, automatization of consistent components may be facilitated during training by allowing trainees to attend fully to the isolated components.

Part-Task Training Again?

Component-based or part-task training is not a new idea. The part-task training literature especially is replete with examples of failed training procedures. Nevertheless, many procedural training programs informally break down the training into parts which are trained individually and then in aggregate. For example, flight instructors often teach students a procedure to scan instruments during flight. This instrument drill is practiced in isolation until the student is comfortable with the procedure. Practice is provided on flight simulators, and students are sometimes encouraged to practice on aircraft in the hangar. By the time the student is actually flying an airplane, the instrument drill is supposed to be "second nature." Under the current perspective, the instrument drill is a task component that should be trained to automaticity so that controlled processing is freed up during flight for aircraft control. Next consider the problem of trying to do calculus without first automatizing basic math and algebra skills. If the algebraic skills are not automatic, they will be unreliable when performed concurrently while allocating controlled processing to performance of the calculus task. The student would be more likely to be error prone, slow, and unable to perform complex problems. For many tasks it is important to automatize key components of the task.

Time-Compressed Training

One of the benefits that falls out of a componential approach to training is the capability of providing a large number of trials for any given component in a relatively short period of time. For example, in an air intercept control training regime for Naval Air Intercept Controllers it is important to be able to visually estimate the angular heading of a radar blip within 5 degrees of accuracy. This level of accuracy takes an average of 2,000 training trials to achieve in laboratory tests. Under normal training conditions, this many trials would require about 5.5 weeks of training time. In a time-compressed angle judgment module, Regian and Schneider (1986) had students perform a video-flash-card version of the task. In this form, students experienced 2,000 trials of the critical task in 3 hours and achieved the requisite accuracy.

AI for Drill and Practice?

Artificial Intelligence programming techniques are, of course, not required for building simple drill-and-practice exercises. The "intelligence" in high-performance training would be manifest in decisions regarding exercise selection and sequencing, decisions rules for when to move from one exercise to the next, and perhaps in real-time generation of specialized drill-and-practice routines. For example, in air intercept control, suppose that the student quickly developed facility with identifying radar blip headings that were near the cardinal points (0°, 90°, 180°, 270°) but was still error prone when identifying other heading angles (e.g., 27). An intelligent system would note this fact and generate drill-and-practice exercises that were heavily weighted with noncardinal practice trials. In most cases, however, tasks are not purely high-performance

or knowledge-rich but rather, involve both kinds of components. Aircraft piloting, for example, involves both skill and knowledge.

INFLITE

The INFLITE system trains students to land a simulated aircraft (F-16) using instruments only. INFLITE runs on AT-class microcomputers. It is written in the C programming language and uses the CLIPS¹ expert system shell. The system was designed primarily to operate with a joystick, but also supports a keypad interface. For optimal utility, INFLITE requires a peripheral voice synthesis device capable of converting an ASCII character stream into articulated speech. Such devices are commonly available and relatively inexpensive. The program presents an accurate real-time simulation of the interactions among essential flight instruments, balancing processor time among the different functional units in order to avoid disruption of the display.

The display interface to INFLITE consists primarily of the HUD (Head-Up Display), which presents speed, altitude, heading, flight path ladder, center indicator, and Instrument Landing System (ILS) beam indicators. In addition to joystick (or keypad) control, the student may toggle on-screen panel lights which depict the status of the landing gear, afterburner, wheelbrake, and airbrake.

INFLITE was designed as an ITS "shell" that will support a variety of instructional approaches including the ability to freeze the simulation to give guidance, prebrief students before training sessions, generate guidance in real time during training sessions, debrief students after training sessions, anticipate student errors in real time based on prior student performance, and generate part-task drills to achieve performance goals. Following is a description of the initial version of INFLITE.

In the initial familiarization session, the student is given a guided tour of the display interface by an articulate coach. As the coach describes each component of the interface, that component is highlighted on the screen. Next, the student is given a series of practice exercises to engender familiarity with methods of controlling the simulation. When the coach is satisfied that the student is sufficiently acclimated, the student is allowed to begin training trials.

At the beginning of each training trial, the simulation commences with the aircraft in flight, under normal flight conditions and randomly positioned with respect to the target airstrip. The goal of each training trial is to successfully land the aircraft. To do this, the student must follow heading information from flight control, use the joystick to turn the aircraft to successive temporary headings, locate the ILS beam, and follow this beam down to the airstrip.

During each training trial, an intelligent coach monitors the simulation (airspeed, heading, deviation from ILS beam, etc.) and provides guidance just as an instructor pilot might guide a student pilot. This guidance is presented verbally, using a voice synthesis system to simulate human speech. The student also receives instructions from a ground-based flight controller, again using synthesized speech but with an alternate voice. During early training trials, the coach may choose to freeze the simulation to give guidance. During all trials, student performance

information is recorded for later use in prebriefing, interactive comments, and postbriefing by the flight coach. During the postflight debriefs, the coach reviews the student's performance in comparison to performance on earlier flights and highlights specific areas to be worked on in future flights. During early training trials, the coach provides guidance and feedback to the student based on real-time observations, as well as by anticipating problems based on typical novice tendencies. In later training trials, the coach additionally anticipates problems based on the current student's performance history. During the preflight briefs, the coach reminds the student of problem areas that were identified in earlier flights.

The intelligent tutoring is handled by the expert, coach, and student modeling modules built with the CLIPS expert system.² The expert outlines suggested pilot actions and judges flight conditions. For example, the expert may suggest a change to a heading of 90 degrees to move the aircraft toward the ILS beam center. If the student does indeed choose to move toward the beam, the expert will note the heading chosen and the effect on the alignment of the craft with the ILS beam. The expert reports to the coach any motion toward or away from the landing goal and its subgoals. The student never hears directly from the expert. Instead, the expert provides information to the coach, who decides how best to interact with the student from an instructional perspective.

The coach uses information such as expert performance data, average student performance data, current student performance data, and the history of the current student to make instructional decisions. For example, the coach may choose to intervene with the student by generating a simple warning ("Wes, you've drifted off course again"), by selecting a part-task drill, or by freezing the display and generating a lengthy description of the problem. In the current version of INFLITE, the display is frozen only in early training trials, such as the first time the student encounters the problem of being aligned with the beam according to the ILS scales but moving away from the beam due to an incorrect heading.

The student modeling module records the student performance profile, which includes: latency and accuracy measures for key task components, common errors (mastered, unmastered, and presented but unlearned), deviations from the expert-suggested actions, short descriptions of the starting flight conditions and the history of the flight, and any coach conclusions.

SUMMARY

Intelligent Tutoring Systems should be sensitive to the full range of performance determinants for their target domain, and should be capable of generating instructional exercises that are appropriate. Many important tasks such as aircraft piloting, complex equipment operation, and electronic troubleshooting involve both knowledge-rich and high-performance components. At the AFHRL, we are investigating ITS architectures that can support the full range of training approaches that are required for these kinds of tasks.

INFLITE is a prototype example of a potential class of intelligent microprocessor-based training simulators with the goal of filling the gap between classroom instruction and expensive simulation time. Such a class of simulators would be useful during initial training and for refresher training. INFLITE is a high-cognitive-fidelity/low-physical-fidelity simulator targeted to

teach key cognitive skills required for high-performance tasks after declarative instruction and prior to high-physical-fidelity simulation instruction.

INFLITE will be used as an experimental testbed for purposes of evaluating the relative training effectiveness of various approaches to automated training of high-performance skills. For example, variations on the system are being developed with additional flight condition and effect simulations, increased student analysis, and increased student-coach interaction initiated by the student pilot. The goals of this work are to extend the range of domains for which ITSs can be applied and to increase the effectiveness of ITSs for knowledge-rich domains. In addition to providing guidance for the development of tutors in high-performance tasks, the principles developed here will apply to intelligent tutoring of high-performance task components within knowledge-rich domains.

Notes

1. The C Language Integrated Production System (CLIPS) was developed by the Artificial Intelligence Section (AIS) of the Mission Planning and Analysis Division (MPAD) at NASA/JSC. INFLITE uses version 4.2, which was developed under joint funding from NASA and USAF.

2. For interested programmers, the following is an example of a CLIPS rule definition in INFLITE. The keyword of the definition is "defrule" for "define rule." The name of the rule follows the keyword, then the verbal description. This rule retracts the suggestion that an action be performed to increase the reading on a gauge. The student could have been flying too low relative to his distance to the airport, and has just made the correction himself. The "?request" line triggers the matching of this rule. The subsequent lines (up through the "= >") grab the gauge reading and test for the suggested correction. If the correction has been made, the right-hand side of the rule is executed, the suggestion is retracted, and a message is printed to the screen.

```
( defrule EndLowValueCorrection
  "Retract a suggested increase in a gauge reading"
  ?request <- ( correctionPerformed ?gauge positive ?magnitude )
  ?gaugestatus <- ( gaugecondition ?gauge tooLow ?minValue )
  ( ?gauge ?curValue )
  ( test ( ?minValue ?curValue))
  = >
  ( fprintout t crlf "LOW Speed correction followed..." crlf )
  ( retract ?gaugestatus )
  ( retract ?request ))
```

BIBLIOGRAPHY

- Anderson, J. R. (1983). *The architecture of cognition*. Cambridge, MA: Harvard University.
- Anderson, J. R., Boyle, F., & Reiser, B. (1985). Intelligent tutoring systems. *Science*, 228, 465-462.
- Bloom, B. S. (1984). The 2-sigma problem: The search for methods of group instruction as effective as one-to-one tutoring. *Educational Research*, 13, 4-16.
- Dede, C., & Swigger, K. (1987). The evolution of instructional design principles for intelligent computer-assisted instruction. *Proceedings of the American Educational Research Association*. San Francisco.
- Fisk, A. D., & Schneider, W. (1983). Category and word search: Generalizing search principles to complex processing. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 9, 177-195.
- Hancock, P. A., & Pierce, J. O. (1984). Toward an attentional theory of performance under stress: Evidence from studies of vigilance in heat and cold. In A. Mital (Ed.), *Trends in ergonomics/human factors I*. New York: North Holland.
- Kyllonen, P. C., & Shute, V. J. (1987). A taxonomy of learning skills. In P. Ackerman, R. Sternberg, & R. Glaser (Eds.), *Learning and individual differences*. San Francisco: Freeman.
- Newell, A., & Rosenbloom, P. (1981). Mechanisms of skill acquisition and the law of practice. In J. R. Anderson (Ed.), *Cognitive skills and their acquisition*. Hillsdale, NJ: Lawrence Erlbaum Associates.
- Regian, J.W., & Schneider, W. (1986, July). *Assessment procedures for predicting and optimizing skill acquisition*. Paper presented at the Educational Testing Service Conference on Diagnostic Monitoring, Princeton, NJ.
- Regian, J. W., & Shute, V. J. (1988). Artificial intelligence in training: The evolution of intelligent tutoring systems. *Proceedings of the Conference on Technology and Training in Education*. Biloxi, MS.
- Ritter, F., & Feurzeig, W. (1988). Teaching real-time tactical thinking. In J. Psotka, L. Massey, & S. Mutter (Eds.), *Intelligent tutoring systems: Lessons learned*. Hillsdale, NJ: Lawrence Erlbaum Associates.
- Schneider, W. (1982). *Automatic/control processing concepts and their implications for the training of skills* (Tech. Rep. No. HARL-ONR-8101). Champaign, IL: University of Illinois, Human Attention Research Laboratory.
- Schneider, W. (1985). Toward a model of attention and the development of automaticity. In M. I. Posner & O. S. Marin (Eds.), *Attention and performance XI* (pp. 475-492). Hillsdale, NJ: Lawrence Erlbaum Associates.

- Schneider, W., Dumais, S., & Shiffrin, R. M. (1984). Automatic and control processing and attention. In R. Parasuraman & D. R. Davies (Eds.), *Varieties of attention* (pp. 1-27). Orlando, FL: Academic Press.
- Schneider, W., & Fisk, A. D. (1982a). Concurrent automatic and controlled visual search: Can processing occur without resource cost? *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 8, 261-278.
- Schneider, W., & Fisk, A. D. (1982b). Degree of consistent training: Improvements in search performance and automatic process development. *Perception & Psychophysics*, 31, 160-168.
- Schneider, W., & Fisk, A. D. (1984). Automatic category search and its transfer. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 10, 1-15.
- Schneider, W., & Shiffrin, R. M. (1977). Controlled and automatic human information processing: I. Detection, search, and attention. *Psychological Review*, 84, 1-66.
- Shiffrin, R. M., & Schneider, W. (1977). Controlled and automatic human information processing: II. Perceptual learning, automatic attending, and a general theory. *Psychological Review*, 84, 127-190.
- Sleeman, D.H., & Brown, J.S. (Eds.). (1982). *Intelligent tutoring systems*. London: Academic Press.
- Soloway, E., & Littman, D. (1986). Evaluating ITSs: The cognitive science perspective. *Proceedings of the Research Planning Forum for Intelligent Tutoring Systems*. San Antonio, TX: Air Force Human Resources Laboratory.
- VanLehn, K. (1986). Student modeling in intelligent teaching systems. *Proceedings of the Research Planning Forum for Intelligent Tutoring Systems*. San Antonio, TX: Air Force Human Resources Laboratory.
- Wenger, E. (1987). *Artificial intelligence and tutoring systems*. Los Altos, CA: Morgan Kaufman.
- Wickens, C. D., Sandry, D., & Vidulich, M. (1983). Compatibility and resource competition between modalities of input, central processing, and output: Testing a model of complex task performance. *Human Factors*, 25, 227-248.
- Woolf, B. P. (1987, June). A survey of intelligent tutoring systems. *Proceedings of the Northeast Artificial Intelligence Consortium*. Blue Mountain Lake, NY.
- Yazdani, M. (1986). Intelligent tutoring systems survey. *Artificial Intelligence Review*, 1, 43-52.

ISSUES IN REPRESENTING KNOWLEDGE
FOR TRAINING
HIGH PERFORMANCE SKILLS*

Pamela K. Fink
Southwest Research Institute
San Antonio, Texas 78284

SUMMARY

How knowledge is represented and used in a computer program is dependent on the task to be performed. The task to be performed, namely the goal of a training system, provides a viewpoint on the generic, primary, or basic knowledge about the domain. For example, in the domain of automobiles, the task of diagnosing malfunctions generates various viewpoints of the knowledge about cars, mechanics, and electronics. Such viewpoints might include a functional representation that allows a mental simulation of how the automobile works, and an experientially-based representation that provides quick condition-action, pattern matching capabilities. Some domains, namely ones that could be called "high performance," require viewpoints on the knowledge that include physical skills. An example is the operation of a console in the Mission Control Center at NASA's Johnson Space Center. Such skills require both a knowledge of the capability which is indicated by accuracy, as well as the ability to perform the task quickly and accurately while doing something else. Representing the accuracy of the task involves the more traditional approaches, while representing the "knowledge" that indicates the speed and automaticity issues is a somewhat novel viewpoint and requires further research. An intelligent tutoring system in the area of mission control console operations is under development that will be capable of training to automaticity a specific task. It is serving as a vehicle for research into how knowledge in high performance domains can be represented and used. It will also be used to research the effectiveness of an intelligent tutoring systems approach to training in high performance domains.

KNOWLEDGE REPRESENTATION

Knowledge representation is one of the major areas of research in artificial intelligence (AI) today. It could be defined as the organization and codification of knowledge, and the placement of the result into a computer program in some kind of computer useable form. What kind of knowledge is organized and codified has changed through the years, depending on the current philosophy for building an intelligent system. Earlier work, during the 1960's, emphasized the representation of general problem solving knowledge. The goal was to keep knowledge about a specific problem area to a minimum and to use powerful universal problem solving techniques that would be capable of solving any type of problem. Later work, during the 1970's and 1980's, has emphasized the representation of more specific, domain-oriented knowledge. Here the object is to fill a computer program with as much knowledge about a particular problem area as possible and to use weaker, but more specific, techniques to solve a specific set of problems. The first paradigm resulted in the development of general purpose problem solvers while the latter has resulted in expert systems.

Support for this research was provided under RICIS Research Activity No. ET.5 (NASA Cooperative Agreement NCC 9-16) through funding from the Air Force Systems Command's Human Resources Laboratory.

The current design principle for intelligent systems, namely "in the knowledge lies the power" [Feigenbaum, circa 1980] meaning that the amount of power or intelligence displayed by a computer system is directly related to the amount of specific, domain-oriented knowledge that is embodied in it, has led to an increase in the importance of knowledge representation to the field of AI over the last fifteen years. The current practical success of the area of AI known as expert systems is a direct result of this pragmatic approach to building intelligent systems. This technology fosters the belief that intelligent behavior can result from the appropriate application of specific knowledge in a given situation, so much work has centered around how to get such specific knowledge into a computer program and applied at the appropriate time during a problem solving task. Today still, the approach is very problem and domain-oriented, which is quite different from the earlier work in general purpose problem solvers. Experience has shown that reasonably intelligent behavior can be obtained by a computer program if the system has all of the specific, detailed knowledge for what to do in all of the situations that it may face. The result has been the development of a multitude of intelligent systems, usually referred to as expert systems, but better described as "idiot-savants." Such systems can be developed to have near expert capabilities in the particular problem solving area for which they have been well-prepared with knowledge, but they break down quickly as the problem moves out of that specific area.

Over the years the research in knowledge representation has resulted in the development of various techniques for representing knowledge, including production rules, frames, semantic networks, and scripts. Each of these techniques is best-suited for representing a particular kind of knowledge. For example, production rules work well for more dynamically-oriented knowledge based on heuristics and condition-action pairs. For more static knowledge, frames work well for representing hierarchies of objects with their attributes and values while semantic networks handle the representation of relationships well. Combinations of these techniques, such as rules that manipulate frames or semantic networks that connect sets of frames into relationships, is also possible. All of these techniques are very good for representing the kinds of knowledge used more or less consciously during a problem solving task. They can represent classes of objects in the world (ie. frames) or chains of reasoning in a particular situation (ie. production rules).

Knowledge representation is an important issue in an intelligent tutoring system (ITS) because it is by this means that the system obtains a knowledge of the domain to be tutored, as well as a knowledge of what the student does and does not know. These portions of an ITS are often referred to as the expert and student modules, respectively. In addition, an ITS uses the knowledge from these two modules, along with some form of teaching knowledge, to determine how to proceed during a training session. Thus, representation of knowledge of various sorts is essential to the development of an ITS.

The following section provides a general discussion on issues involved with representing domain knowledge in an ITS. Then the problems involved specifically with training in a type of domain referred to as "high performance" are described as well as the hierarchy of knowledge that must be trained in order for a student to proceed from novice to expert in a high performance domain. Finally, the issues involved with representing and using such knowledge in an ITS to train will be analyzed, followed by a discussion of a particular system that implements these ideas for intelligent tutoring of a high performance domain, namely propulsion console operations in the Mission Control Center at NASA's Johnson Space Center.

REPRESENTING DOMAIN KNOWLEDGE IN AN INTELLIGENT TUTORING SYSTEM

Though some work is again underway to represent generic domain knowledge independent of the task to be performed [Porter et al., this volume], most representations center around the development of an appropriate task-oriented viewpoint of the knowledge. This same philosophy is reflected even in the definition given by Waterman in his book A Guide to Expert Systems for the term "representation", which he defines as "the process of formulating or viewing a problem so it will be easy to solve" [Waterman, 1986, p. 11]. Thus, the selection of one or more of the various knowledge representation techniques for representing the knowledge in a given situation depends on the task to be performed as well as the inherent structure and nature of the domain. The task to be performed, namely the goal of a training system, provides a viewpoint on the generic, primary, or basic knowledge about the domain. For example, in the domain of automobiles, the task of diagnosing malfunctions generates various viewpoints of the knowledge about cars, mechanics, and electronics. Such viewpoints might include a functional representation that allows a mental simulation of how the automobile works, and an experientially-based representation that provides quick condition-action, pattern matching capabilities. Figure 1 illustrates this notion of a task-oriented viewpoint that creates a projection onto the more generic, task-independent version of the knowledge.

One would hope that work in ITS's could benefit from all of the research and experience that has accumulated in knowledge representation and expert systems. At first it seems reasonable to assume that an expert system for a particular task should be useful as the expert module in an ITS. This is unfortunately not always the case for various reasons, a major one being that an expert system developed without tutoring in mind will most likely not embody the knowledge to be trained in an appropriate format for training. An expert system may be able to solve a problem but not necessarily in the way that one would wish to train people to do it. Some work has been done in the area of representing and using knowledge in a way similar to human experts for certain training tasks such as diagnosis [Smith et al., 1985, for example] but much work still remains.

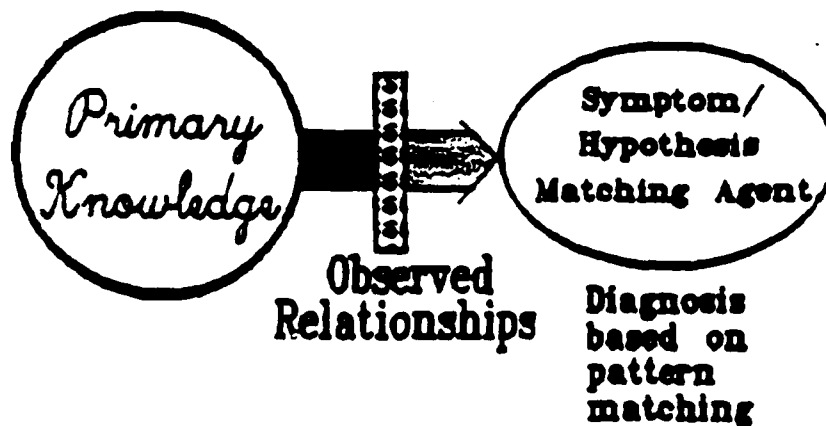


Figure 1. Creating a Knowledge Projection by Using Experience to Enforce Structure on the Primary Knowledge of a Given System

Furthermore, the knowledge representation techniques used in expert systems are not always well-suited for representing the "knowledge" required for training certain types of tasks, such as those involved with the performance of a certain skill. These kinds of domains require a viewpoint on the knowledge that includes physical skills, so one would not tend to build an expert system to perform the task in the first place. These skills require both a knowledge of the capability which is indicated by accuracy, as well as the ability to perform the task quickly and accurately while doing something else. Representing the accuracy of the task involves the more traditional approaches in knowledge representation, while representing the "knowledge" that indicates speed and automaticity issues is a somewhat novel viewpoint. To perform a particular skill initially does require a certain amount of knowledge and some conscious thought when using that knowledge. However, as an individual solves more and more problems and gets better and better at a given task, this knowledge somehow becomes easier to use. An experienced expert, especially in a performance-based task, is much quicker at finding and using the appropriate knowledge to solve a particular problem. Through experience, the more general, novice-type knowledge gets reshaped into more specific, expert-type knowledge while not losing its original more fundamental useability. This transition from novice to expert in a human during learning is an important issue for knowledge representation in an intelligent tutoring system. It is of particular importance in the training of performance skills where the original knowledge is no longer even used, at least consciously, to perform a given task. Such domains are referred to as "high performance" domains [Regian and Shute, 1988].

Most of the intelligent tutoring research to date has focussed on tasks from knowledge-rich domains. For example, Anderson's Lisp Tutor [Anderson, Farrell, and Sauers, 1984], Brown and Burton's SOPHIE system for electronic diagnosis [Brown, Burton, and deKleer, 1982], Carbonell and Collins' SCHOLAR system for South American geography [Carbonell, 1970], and Woolf and McDonald's MENO-TUTOR for diagnosing non-syntactic bugs in computer programs [Woolf and McDonald, 1985] all deal with domains that emphasize the conscious use of knowledge. Such tutoring systems attempt to impart certain static and/or procedural knowledge that the student is then tested on as much for the knowledge content, as for the problem solving skill.

Tasks which are primarily performance-based have not attracted much intelligent tutoring research attention to date. Tutoring systems in these domains must be capable of imparting not only a certain amount of knowledge but also of drilling the student in the use of this knowledge to the point where the student need no longer concentrate on the actual problem solving task. At this point, the task is "automatized," and the individual is free to concentrate on other, more cognitively demanding issues while still performing the trained task. Testing to determine if an individual has a particular piece of knowledge is quite different from testing to determine if he/she has automatized a particular skill based on that knowledge. How such knowledge and skills should be trained also varies greatly from the approaches used in more traditional ITS domains.

TRAINING IN A HIGH PERFORMANCE DOMAIN

Acquiring the ability to perform a particular skill usually requires several stages. For example, a student learning to play the piano must first assimilate a certain amount of fairly static knowledge such as what notes are associated with what keys on the keyboard, how to read the notes written in a musical score, understand the meaning of various terms used to provide guidance in interpreting the written notes, etc. The student is also usually presented with some exercises, such as scales and chords, to be repeated over and over that familiarize him or her with the actual process of playing the piano. The piano student must then learn to convert all of this knowledge into actions of the hands and fingers to generate the actual music. This is usually accomplished

through hours and hours of practice, both directed by the teacher as well as on the student's own. Eventually, after a certain amount of practice that usually varies with the student and the type of music being played, the student no longer needs to think about reading the notes and moving the fingers. Playing the instrument has been automatized and the student can now spend cognitive effort on other things, such as singing.

This process of learning a high performance skill can be broken down into several distinct stages:

1. static overview knowledge, where general background information is provided
2. general procedure-oriented knowledge, where the general steps in performing the task are presented
3. guided-example exercises, where specific examples are given and the student is provided the opportunity to practice while being prompted and coached in order to develop accuracy in the skill
4. unguided-example exercises, where specific examples are given and the student is provided the opportunity to practice the whole process without interruption in order to develop speed with accuracy
5. automated-example exercises, where specific examples are given and the student is provided the opportunity to practice the whole process while doing another task in order to develop an automated capability

In a domain such as the operation of a particular piece of equipment, these steps correspond to

1. a general system overview that describes the salient parts and features of the particular piece of equipment on which the task will be performed (a "guided tour")
2. a description of the steps that must be performed in executing the particular procedure to be trained, such as "initialization" or "self-test," indicating the parts of the equipment involved and the motivation for each step and/or its effect on the overall status of the goal
3. presentation of specific example cases where the student must perform the procedure on the equipment, or a simulation of it, in order to solve the problem, but with the instructor watching and guiding as necessary
4. independent drill and practice where the student must perform the procedure on the equipment, or a simulation of it, with no guidance during actual exercise performance but only a final evaluation of accuracy and speed in order to become proficient at the procedure
5. independent drill and practice where the student must perform the procedure on the equipment, or a simulation of it, while performing another task so that performing the trained procedure becomes automatic and conscious thought can turn to the performance of an additional task

This training process moves the student gradually through the various phases of skill development. When to move on to the next level of training versus when to remain at the same level or even backup and remediate, varies with each student. Also, the

scale for defining the quality of skill performance varies with the student. Some students may never perform as well on the task as other students. Thus, the system must be capable of recognizing when a student has "peaked" on a particular training phase. These are issues where certain AI-based techniques could be used to provide the training system with the needed knowledge to make these decisions on a case-by-case basis, thus moving toward a more adaptive, responsive, intelligent tutoring system.

REPRESENTING DOMAIN KNOWLEDGE FOR TRAINING IN A HIGH PERFORMANCE DOMAIN

As illustrated in the previous section, high performance skills could be taught in five distinct phases, each associated with a specific kind of knowledge and a particular means for teaching and testing it. The following paragraphs describe these issues for each of the five levels. The types of learning strategies that are recommended at each phase of the training were selected from the list that appears in [Kyllonen and Shute, 1988, Table 1, p. 15].

The static overview knowledge taught in phase one consists of the general, enabling knowledge concerned with the skill to be performed. It provides the "big picture" view of the objects and their relationships to one another in the domain that are important to performing the task. For example, a system overview should be organized from the viewpoint that is desirable for performing the particular task and not necessarily include everything there is to know about the equipment. Thus, representation is fairly straightforward since frames and semantic networks work well for such types of knowledge. If training is oriented towards the operation of a particular piece of equipment, then the system would have a knowledge of all of the important pieces and components, their name, their description, their function within the system as a whole, etc. An appropriate learning strategy for this particular kind of knowledge is simply rote memorization. The student needs to assimilate all of the facts and relationships with little need for further exploration and no need for modification. He/she needs to be capable of identifying components based on appearance, location, and any other attributes salient to the task to be trained. An ITS can simply present the information and then test it in some kind of static format such as multiple choice, fill-in-the-blank, or identification. The student is assumed to know the material when he/she can answer all of the test questions correctly.

The teaching of the initial, general procedural information is also involved with fairly static knowledge. Each step can be presented in the order in which it should occur, along with the reasons and motivations for performing that particular step at that particular point in time. Though the knowledge is about a procedure, it is static in nature. A directed graph format is appropriate in such a situation because it is capable of representing the sequential nature of the knowledge. The nodes of the graph correspond to frames that represent the steps of the procedure while the arcs indicate the legal transitions between steps. An appropriate learning strategy for this knowledge is again rote memorization with possibly some learning from instruction. Though what is presented could be learned exactly as presented, a student may choose to internalize the procedure in a slightly different way than it is actually presented by the tutoring system, especially at the beginning of training when all conditions and exceptions that govern the sequence of steps in the procedure are not apparent. Testing can again use such techniques as multiple choice, fill-in-the-blanks, and identification to ensure that the student has assimilated enough of the general knowledge to move on to the next step. The student is assumed to know the material when he/she can answer all of the questions correctly.

The third phase of the training process, guided example, begins to move into actual skill acquisition. In this phase the student is presented with some specific examples of

procedures to be performed and is guided, or coached, through them. The goal of this phase is to provide the student with enough experience in actually performing the procedure to acquire the ability to do it fairly accurately. However, in order to prompt the student and coach him/her through the performance of a task during this phase of training requires a considerable amount of knowledge. First, the system must itself know the procedure and how to perform correctly on the specific exercise. This knowledge is more or less provided by the directed graph representation of the procedure developed for phase two of the training. Second, it must not only be able to identify when a student is performing the task incorrectly but also be able to infer why the student has made the error so that it can provide appropriate feedback. This can be quite difficult since the only source of information that the system has available concerning what the student does and does not know is deduced through the student's input to the system during the exercise, along with information on how long it took for the student to respond and the student's performance on previous exercises.

If the system has a certain understanding of the general structure of the procedure to be performed, certain mistakes that a student makes can be interpreted to indicate certain errors or misconceptions. For example, if the student performs a step that happens to be one of the steps that is legal to perform directly after the current legal step, then the system could assume that the student simply skipped a step, for whatever reason. We call this a "simple step omission," meaning that the student has just forgotten a step in the procedure. However, if the student performs a step that is several legal actions away from the current one it could mean one of several things. First, if the steps that have been skipped are all logically grouped under a particular phase of the procedure, the system could assume that the student simply forgot a higher level "step" in the procedure --- one that happens to consist of several separate actions. We could call this a "simple group omission" in the same manner as when a single step is omitted since logically only a single step was skipped. However, if there is no relationship between the steps skipped, it is possible that the student is lost to some degree and simply remembered that that step has to be performed at some point and therefore performed it for lack of a better idea of what to do next. We could call this a "complex omission" meaning that the student probably does not yet understand the fundamental steps involved in performing the procedure. A final type of error that a student could commit is to perform an action that is not legal at any point in the current procedure. Then one might assume that the student has remembered the procedure wrong, has confused it with another procedure, or is completely lost. We call this an "insertion error."

All of these evaluations of what the student might have been thinking that would cause him/her to perform the procedure incorrectly are based initially on the student making this error for the first time. However, if within a procedure a student consistently makes the same mistake(s), the system could assume that the student can not remember what to do next and the system will begin to provide hints to the student to help him/her figure out what to do next. If the student continues to make errors, the system will eventually tell the student specifically what is expected to be done next. If failures still occur with little improvement on the part of the student, the system will decide to provide some remediation, backing up first to again demonstrate specific examples and if problems still occur, eventually backing up to the previous phase of training in order to provide more basic remediation.

The type of learning strategy applied here is drill and practice. The student can refine and tune his/her knowledge through applying it in a variety of situations through the assignment of exercises to be performed. Testing in this phase involves determining if the student can perform a given exercise with enough accuracy and reasonable speed to indicate that he/she has actually learned the procedure and knows what to do at each step. The number of trials required to attain proficiency can vary greatly from student

to student, as can the ultimate level of proficiency. The goal is to determine what the student knows based on actual performance, not on how many times they have done it. This requires a decision on how accurate is accurate enough and how fast is fast enough at this level to indicate that the student is ready to move on to the next phase. Another question is how many times must an individual demonstrate the needed accuracy and speed before the system can assume that the student can consistently sustain this level of performance.

The fourth phase of the training process, unguided examples, requires the student to perform example exercises without guidance or support from the tutoring system. The only feedback that is provided occurs once the student indicates that he/she has completed a given exercise. Then the system can provide an assessment of the student's performance in terms of accuracy and speed. The goal of this phase is to provide the student with enough experience in performing the procedure straight through without interruption to acquire the needed speed while maintaining accuracy. The knowledge required to evaluate student performance in this phase is not as extensive as that required during the guided examples phase because the system is no longer trying to catch and interpret student errors as they occur. The system still needs to know how to perform the procedure and to identify when errors have occurred, but this identification occurs too late to help guide the student back onto the right track. Accuracy is more or less assumed in this phase and speed of performance is the focus. It is at this phase that the problem of how to represent skill rather than cognitive knowledge becomes an issue. Accuracy in many cases will have to be absolutely correct, but the acceptable speed may vary a lot, depending on the actual domain as well as the student. Thus, representation of skill must be a sliding scale that evaluates both accuracy and speed with respect to the domain of application as well as the perceived potential capabilities of the student.

The type of learning strategy applied in this phase is again drill and practice. The student can refine and tune his/her skill through repeatedly applying his/her knowledge to perform the requested procedure. Testing in this phase involves determining if the student can perform a given exercise with increased speed while maintaining accuracy to indicate that he/she has acquired a certain level of proficiency in performing this procedure. As in the previous training phase, the number of trials required to attain a given skill level can vary greatly from student to student. The goal is to determine how skillful the student is based on actual performance, not on how many times they have practiced. This requires a decision on how accurate is accurate enough and how fast is fast enough at this level to indicate that the student is ready to move on to the next phase. Different students will most likely have different peak levels of performance that they can attain and sustain, especially in terms of speed.

The fifth and final phase of training provides the student with exercises that will cause him/her to automate the process, meaning that the student will become capable of performing another task that requires cognitive processing while maintaining speed and accuracy in the initial procedural task. Here the student is presented the same kind of exercise as in the previous two phases of training but in addition must also perform some simple but new task, such as hitting a particular function key on the computer keyboard based on the pattern of tones heard. The goal is to bring the student to the point where he/she no longer has to think about how to perform the procedure being trained. The student knows the procedure so well that cognitive effort is no longer required to perform it. At this point the skill has been automatized. How this skill level is best represented to the computer so that when a student reaches this point the system can recognize it, is somewhat of a research question. Currently, we are working on expansions of the notions already used in the previous stages such as accuracy and speed. In this phase, however, these must be used for judging student performance in both tasks at the same time.

The type of learning strategy applied in this phase is again drill and practice. The student can refine and tune his/her skill through practice in a variety of situations. Testing in this phase involves determining if the student can perform a given exercise maintaining accuracy and speed while performing another task accurately to indicate that he/she has acquired a certain level of automaticity in performing this procedure. As in the two previous phases of the training process involved specifically with skill acquisition, the number of trials required to attain a given skill level can vary greatly from student to student. The goal is to determine how skillful the student is based on actual performance, not on how many times they have done the procedure. This requires a decision on how accurate is accurate enough and how fast is fast enough at this level to indicate that the student has automatized the procedure and has, therefore, attained a level sufficient to conclude training. Again, different students will most likely have different peak levels of performance that they are capable of attaining.

How long a student might spend at any given level of training is, of course, unknown. It will depend to some degree on the difficulty of the domain as well as the individual student's capabilities. No set number of trials, level of accuracy, or rate of performance improvement is predetermined. These can be varied based on student aptitude and domain complexity. This type of training could be useful in screening students for aptitude in a particular type of skill by identifying those that develop high levels of accuracy and speed with a minimal number of trials.

AN EXAMPLE SYSTEM

An intelligent tutoring system for a high performance domain is currently under development. The area of application is the operation of consoles in the Mission Control Center at Johnson Space Center. An example of such a console, namely the front-room propulsion console, is presented graphically in Figure 2. These consoles vary somewhat from one function to another, but they generally consist of:

1. one or more video displays
2. numerous sets of indicator lights, referred to as Display Decoder Drive Event Lights (DDD lights)
3. various manual entry devices consisting of numeric thumbwheels and push button indicators including the voice keyset, the manual select keyboard (MSK), the summary message enable keyboard (SMEK), and the display request keyboard (DRK)

as well as one or more other panels for displaying various times associated with the mission. These consoles may also be attached to one or more strip chart recorders for recording sets of analog signals.

In order to become proficient at operating such consoles, flight controllers must learn how to

1. format the various DDD light panels using the MSK
2. select, display, and read a variety of video display formats using the MSK, SMEK, and DRK
3. select and listen to various voice loops using the voice keyset

PROP CONSOLE

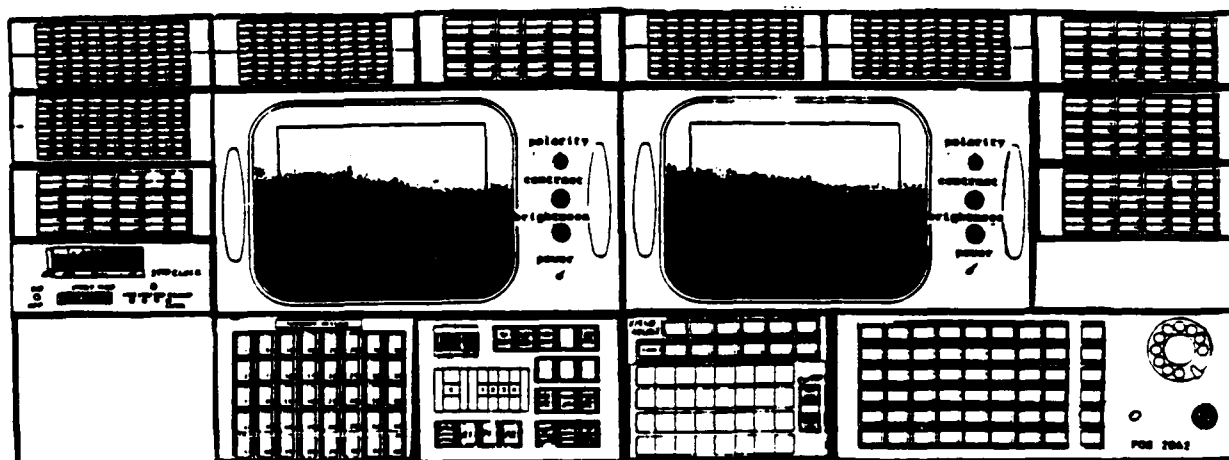


Figure 2. A Sketch of the Propulsion Console
in the Mission Control Center at Johnson Space Center

as well as many other things. They must learn to operate these consoles in an automatic manner because such operations are only a means for achieving another goal, namely ensuring the safe and correct operation of a particular system, such as propulsion, during a mission. Should a situation arise where data must be accessed, analyzed, and interpreted, the flight controller must be capable of quickly and effectively accessing the needed data from various video displays and DDD lights without specific thought as to how to manipulate the various keyboards. Their conscious, cognitive thoughts are too busy dealing with the situation to be concerned with how to get the data. Thus, console operations can be classified as a high performance domain.

Current work on the intelligent tutoring system for console operations has centered around training the operation of the Manual Select Keyboard (MSK). This is the keyboard that is used during initialization of the console for the ascent, orbit, and descent, phases of a mission. Initialization requires the formatting of all DDD light panels, the selection of several video displays to get information concerning general system status, and the selection of various voice loops to listen in on appropriate monologues and dialogues. Eventually the tutoring system could be expanded to include training on all of the various components of a console, as well as a general console overview. Figure 3 provides a hierarchical, graphic representation of the knowledge needed to effectively operate a Mission Control Console. Figure 4 provides a detail of the five-phase training methodology for the MSK.

The display for the tutoring system is organized into three major windows, as illustrated in Figure 5. Across the top third of the screen is a complete graphic representation of the entire console. This provides the student with an overall layout and organization of the console. The lower left half of the display provides an area where one of the panels from the console can be expanded to provide further detail. The figure shows the MSK panel. The lower right half of the screen provides the text interface where the tutor can present information, exercises, and accept student responses to specific verbal questions.

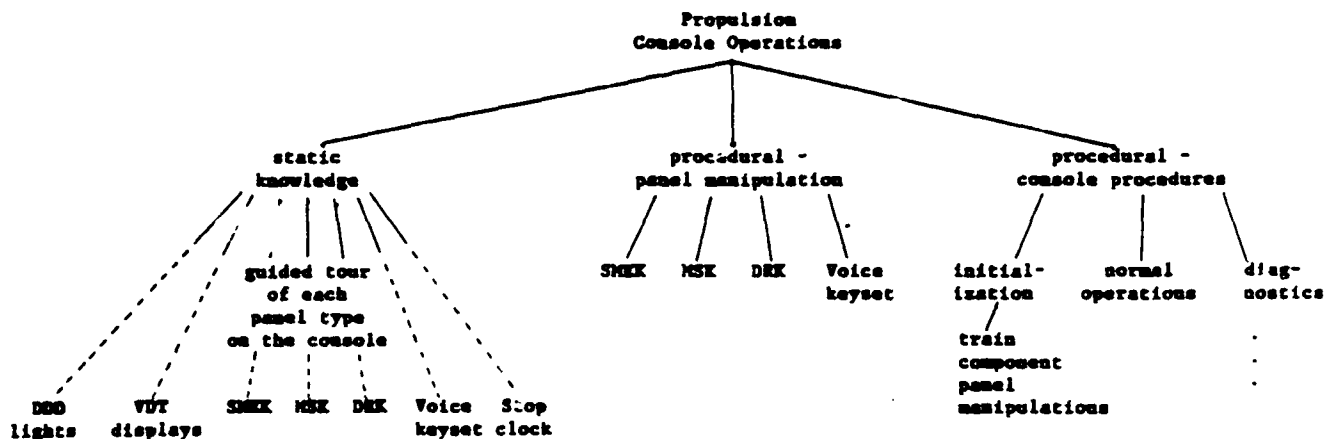


Figure 3. Top Level Overview of the Knowledge Needed to Train Console Operations

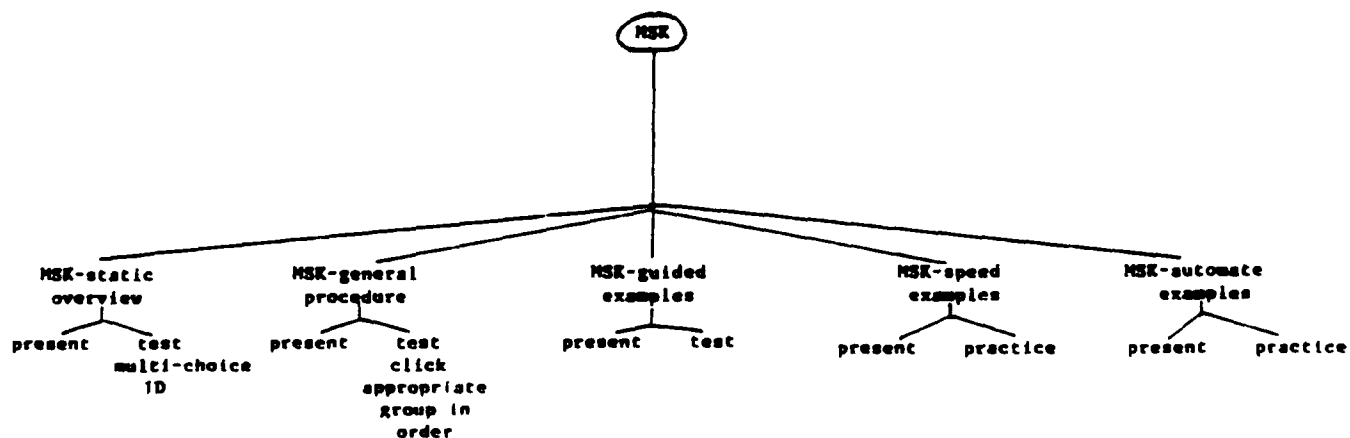
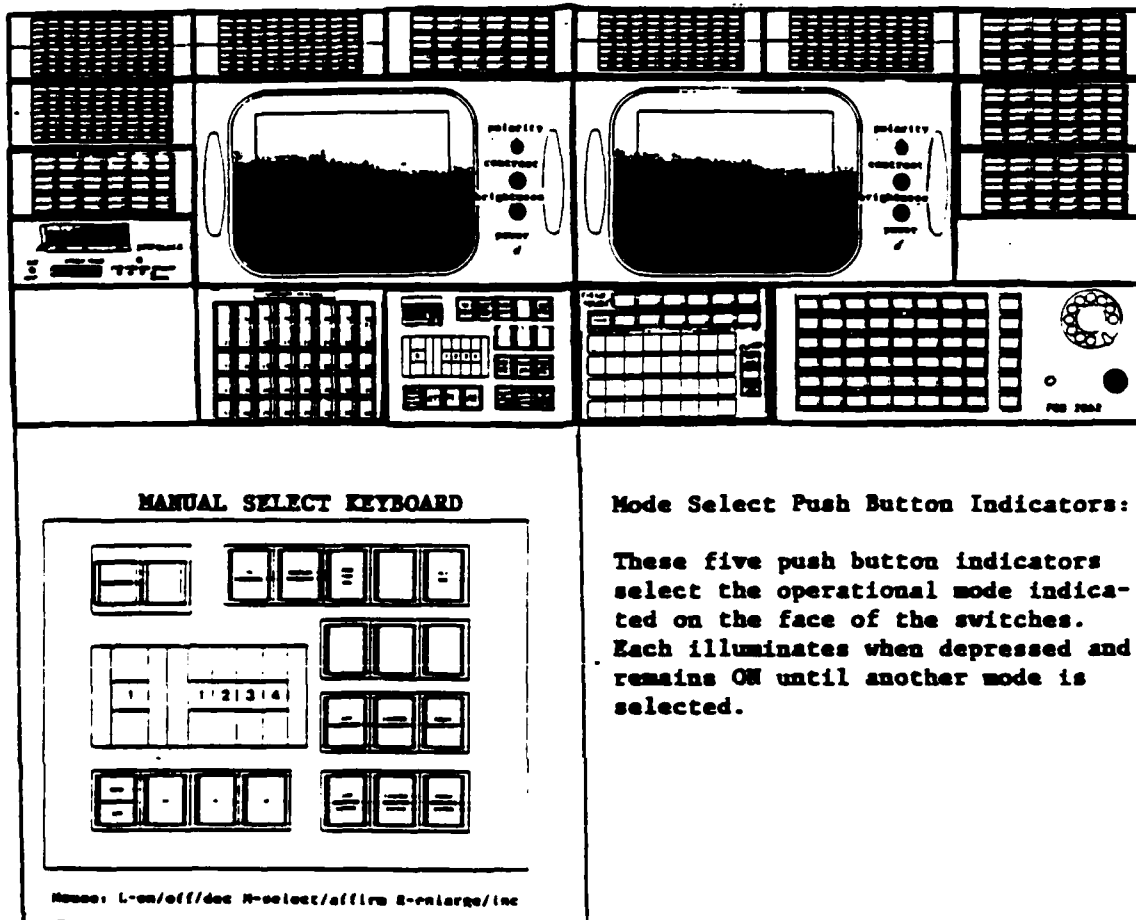


Figure 4. Overview of the Knowledge Needed to Train the Use of the Manual Select Keyboard (MSK)



**Figure 5. Layout of the Tutoring System's
Graphic Interface with the Student**

The graphic display of the console is mouse-sensitive. Under certain conditions it allows the student to select panels by clicking over them with the mouse to have them blown-up in the lower left window of the display. When a panel is expanded and displayed in the lower left window, it too is mouse-sensitive. A student can manipulate it by clicking the mouse over its components, thus incrementing or decrementing a thumbwheel counter, turning a push button indicator on or off, or just getting a display of the text written on the object. In this way, a large portion of the console functionality is simulated graphically and the student can gain experience in performing console operations through these simulated manipulations.

Training on the use of the MSK proceeds through the five phases described in the previous section. The first phase of training on the MSK provides an overview of what the MSK consists of. The MSK panel is blown-up in the lower left window on the screen and the system steps through each of its functional components, highlighting them on the graphics display and describing them with text in the lower right window. This is illustrated in Figure 5, where the mode select push button indicators are highlighted in the graphics on the left and their description appears in the text on the right. A student can move forward and back at their own pace through this portion of the tutorial. At the end, the student must pass an identification test where the student is asked to click over the various components of the console to indicate his/her response to the

tutoring system's questions in order to proceed on to the next phase of the training. Based on the score the student is allowed to move on or required to review the material.

An overview of the general procedural process for manipulating the MSK comes next. This is done in a manner similar to the MSK overview. Components are highlighted in order and explanations about each step of the procedure are provided. The procedure for manipulating the MSK varies depending on the mode selected with the push button indicators in the upper right corner of the MSK panel. For example, to request a particular video display to appear in the right monitor of the console, the push button indicator with DISPLAY REQUEST written on it must be pushed, the number of the display entered on the right four thumbwheels, and the RIGHT MONITOR ENTER push button indicator pressed in the lower right corner of the MSK. A number of other steps must be performed as well, but these are the key steps. The order of all steps does not matter, with the exception of pressing the monitor enter push button indicator, which must be done last. Thus, the procedure could be represented as in Figure 6, where five actions comprise an unordered group that constitutes the first step, called "set-up," and the monitor enter action comprises a second step, called "execute." Of course, some sequences are more logical than others and the current system enforces a specific order for performing these actions. In order to move on to the next phase of training, the student must identify each general step in the correct sequence.

In the third phase of the training, specific examples of the procedure are generated that the system solves as a demonstration for the student. Then the student is given an exercise that he/she must do by manipulating the mouse over the appropriate components of the MSK in the correct order to perform the requested operation. For example, the system may request that the student cause the video display number 12 to appear in the left display monitor. The system will prompt the student at each step and verify its correctness before moving on to the next step. If an error occurs, rules like those discussed in the previous section on the guided example phase are used to coach the student to perform the procedure correctly. Satisfactory performance in this phase of training is based mainly on accuracy, but speed is also considered. When a student has consistently performed the assigned exercises completely correctly and the speed of performance has more or less plateaued, then the system allows the student to move on to the next phase of training.

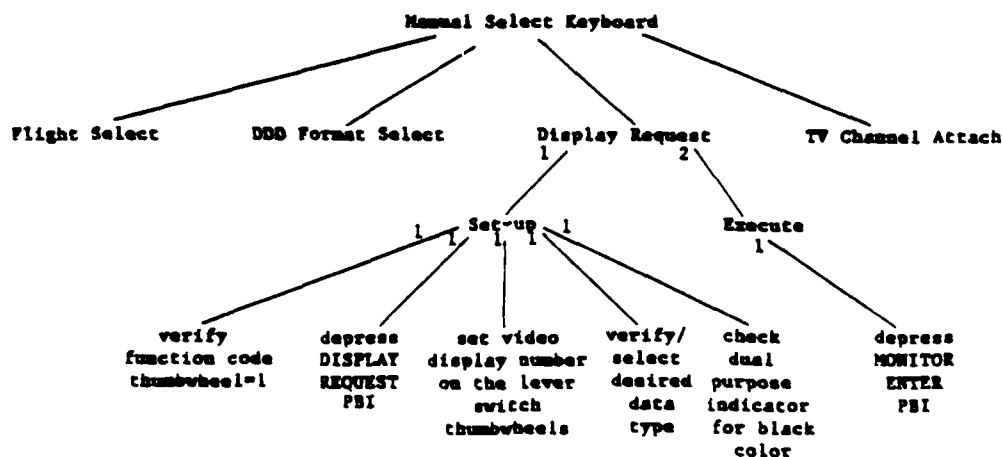


Figure 6. Overview of the DISPLAY REQUEST Procedure Using the MSK

The fourth phase of training no longer guides or coaches the student through the exercises. Instead the system simply presents an exercise, in the same manner as in the previous phase, and the student must manipulate the MSK appropriately with the mouse to achieve the requested action. Based on consistently performing with complete accuracy and reaching a point where speed is no longer improving much, the system then allows the student to move on to the final phase of the training.

The final phase is a repeat of the fourth phase only with an additional task that must be performed simultaneously by the student while doing the assigned exercise. While requesting a particular video display or formatting a set of DDD lights, the student must also acknowledge certain patterns of beeps by hitting the appropriate function key. The system assumes that the student has successfully automatized the MSK manipulation process when the accuracy in performing both tasks has reached one hundred percent and the speed of performing the assigned exercise and responding to the beeps has reached a peak for that particular student.

It is important to note that during the final three phases of training, where skill is being acquired and tested, no predetermined number of trials is used to determine whether or not the student should move on. Advancement to the next phase in training depends on the particular student's performance. Though accuracy is required to be one hundred percent correct, ultimate speed can vary based on the student. The system looks for where the student seems to be leveling off in order to determine when to move on. The decision to backup and review material is based on how much difficulty the student is having attaining the required perfect accuracy. Remediation can backup all the way to the start of the training program if necessary. In this way the system can be used to refresh the memories of individuals who have been interrupted in their training for a period of time, as well as those who are seeing the material for the first time.

CURRENT STATUS AND FUTURE WORK

The intelligent tutoring system for Mission Control Center Propulsion Console Operations is written in C, CLIPS, and GPR and runs on an Apollo Domain 4000 with a color monitor. It is currently capable of training the first three phases of the MSK. The last two phases of training will be working soon. At that time, an experiment will be run to test the effectiveness of such a training system. One goal of the research is to determine if the flexibility provided by an intelligent tutoring system increases the speed with which an individual can automatize a particular process and if so, if the increase in training efficiency will offset the cost for developing such systems. Another question to be answered is how effectively does such training transfer to the operation of the real equipment.

REFERENCES

- Anderson, J., Farrell, R., and Sauers, R., 1984, "Learning to Program in LISP," Cognitive Science, 8, pp. 87-129.
- Brown, J.S., Burton, R.R., and deKleer, J., 1982, "Knowledge Engineering and Pedagogical Techniques in SOPHIE I, II, and III," in D. Sleeman and J.S. Brown (eds.), Intelligent Tutoring Systems, London: Academic Press.
- Carbonell, J.R., 1970, "AI in CAI: An Artificial Intelligence Approach to Computer-Aided Instruction," IEEE Transactions on Man-Machine Systems, MMS-11(4), pp. 190-202.
- Feigenbaum, E., circa 1980, general comment about the AI field.

- Kyllonen, P.C. and Shute, V.J., 1988, "A Taxonomy of Learning Skills," in P. Ackerman, R. Sternberg, and R. Glaser (eds.), Learning and Individual Differences, San Francisco: Freeman.
- Porter, B., Acker, L., Lester, J., and Souther, A., 1988, "Generating Explanations in an Intelligent Tutor Designed to Teach Fundamental Knowledge", this volume.
- Regian, J.W, and Shute, V.J., 1988, "Ai in Training: The Evolution of Intelligent Tutoring Systems," Proceedings of the Conference on Technology and Training in Education, Buloxi, MI.
- Schneider, W., and Shiffrin, R.M., 1977, "Controlled and Automatic Human Information Processing: I. Detection, Search, and Attention," Psychological Review, Vol. 84, pp. 1-66.
- Smith, H., Fink, P., and Lusth, J., 1985, "Intelligent Tutoring Using the Integrated Diagnostic Model: An Expert System for Diagnosis and Repair," in Proceedings of Expert Systems in Government, pp. 128-134.
- Waterman, D., A Guide to Expert Systems, Addison-Wesley, 1986, p. 11.
- Woolf, B., and McDonald, D.D., 1985, "Building a Computer Tutor: Design Issues," AEDS Monitor, 23(9-10), pp. 10-18.

INTERFACE ARCHITECTURES FOR INTELLIGENT TUTORING SYSTEMS

Jeffrey G. Bonar
Virtual Machine Corporation
800 Vinial Street
Pittsburgh, Pennsylvania 15212

SUMMARY

The design of current ITS is closely tied to how those ITS present the domain to a student, that is, to the tutor's interface. Not only is domain knowledge represented in terms of the interface, but so are teaching and diagnostic knowledge. From this observation we propose to explicitly design ITS based on the interface to the task actually presented to the student. This proposal has the potential give the learner a much richer view of their task: a view filtered through the eyes of the expert. Furthermore, this proposal can break the ITS knowledge engineering bottleneck by getting the AI wizard out of the loop and allowing skilled instructors to be putting the knowledge in an ITS system. The paper discusses a working example of this approach, a proposals for a general architecture, and particular version of that general architecture.

THE CENTRAL ROLE OF INTERFACES IN CURRENT INTELLIGENT TUTORING SYSTEMS

Intelligent tutoring systems for non-trivial curriculums organize their knowledge around the presentation of that knowledge to the student. That is, the tutor's internal architecture and knowledge base are specifically designed to support a particular style of interaction with the student. Consider the following examples:

Proust [6]

A tutor for beginning programmers, Proust takes a student program and analyses that program with an intelligent parser that looks for buggy or correct implementations of standard schemas (called "plans"). The knowledge base of Proust is a large catalog of descriptions for correct and buggy implementations of the standard tasks.

Lisp Tutor [11]

Another tutor for beginning programmers, this tutor coaches a student through the specific steps in coding particular Lisp functions. The knowledge base consists of a rule set that accomplishes the task given to the student. There are also rules that simulate common student errors. As a student works, each of his or her steps is associated with a correct or errorful rule.

Geometry Tutor [1]

A tutor to teach geometry proofs. The student constructs a proof tree on the computer screen, where each component (subtree) of that tree places a conclusion as the parent node above the elements that produced that conclusion. The knowledge base of the program specifies the axioms and theorems that allow a student to assemble these components.

Bridge [3]

A tutor for novice programming. The student assembles programming schemas (called "plans") at various levels of detail. The knowledge base of the program consists of plan descriptions and descriptions of how those plans are combined into solutions for various problems.

Intelligent Maintenance Training System [15]

A tutor to teach troubleshooting of complex systems. Systems are represented with screen-based simulations containing faults that must be discovered by the student. The knowledge base is represented in terms of the components of the simulation including both normal and fault states for those components.

In each of the cases cited above, the knowledge base is organized around the particular style of presentation and pedagogy used by the tutor. That is, around the tutor's interface. Furthermore, it is not just the domain knowledge that is organized based on the interface, but also the teaching knowledge and diagnostic knowledge. There is little or no knowledge about teaching or diagnosis that is not specifically tied to the tutor's interface. Practical ITS seem to require careful crafting of knowledge around a particular interface approach. In each of the examples listed above, the tutor detected student bugs, inferred underlying errors, and made corrections, but it did so within an elaborate and sophisticated interface, carefully designed to highlight and draw attention to the task the tutor was designed to teach.

Using Expert Interfaces to Guide ITS Development

The above may seem obvious; of course an ITS needs a well-crafted interface for the student. What I am suggesting in this paper, however, is that beginning with such an interface is the appropriate way to design ITS. Instead of organizing the tutor around a formal representation of the knowledge to be presented - a often touted but as yet not realized goal of ITS research - we should explicitly organize the tutor around how it will present things to the student. In particular, I am suggesting that we organize an ITS around the kind of interface that an expert uses.

When discussing the interface to an ITS, it is important to realize that I mean more than what is on the screen. The interface gives the learner access to the conceptual vocabulary used by experts when they talk about and solve problems within the domain. This vocabulary is a crucial part of the expertise that must be imparted to the student. The right vocabulary allows an expert to cut up a problem into the right pieces, and organizes those pieces into the right order for a solution. This vocabulary need not be words: programmers have flow diagrams, electrical engineers have circuit schematics, etc. I am suggesting that the experts vocabulary, diagrams, conventions, etc. - that is, the interface - can be enormously heuristic in telling us what must be acquired by novices. If we can build ITS around the interfaces used by experts, not only have we focussed on the key issues of concern for the novice, we have also made the acquisition of expert knowledge much simpler because the ITS is designed around how the expert views the knowledge, not around how an AI expert might view that knowledge.

WHAT IS IN AN INTERFACE-BASED ITS?

In this section I discuss the kinds of knowledge in an interface-based ITS.

Three Kinds of Expertise

What is the expertise that we want to capture in an ITS, and how does focussing on the expert's interface help us to do that knowledge capture? We have identified three kinds of expertise that need to be captured. This "three component" analysis of expertise derives from a body of cognitive science research studying learning in semi-formal domains like basic math, physics, computer programming, etc (see Resnick & Omanson [12]). The three components are as follows:

1. **Concepts.** These are the abstract conceptual elements known by someone expert in a field. These are the ideas with which an experts thinks about tasks in his or her domain. Resnick & Omanson [12] points out that in mathematics, even beginners must reason about objects that exist only as abstractions. One can, for example, point to 3 chairs, or even the the numeral 3, but there is no "number 3" that can be pointed to. *Number* is an abstract conceptual element. In computer programming a "loop" is similarly abstract, as is "voltage" in electrical circuits and "force" in mechanics.

Traditionally, curriculums are organized around the attainment of such abstract concepts - e.g. a chapter on each concept or on a related group of concepts. The next two elements focus on aspects of expertise often unrecognized in formal curriculums.

2. **Representations.** These are the syntax and mechanisms used by an expert. They may be embodied in notations, standard forms, standard procedures, or algorithms. These are mechanical in nature, providing efficiency of operation while sacrificing some generality or formal properties. In mathematics the notational system consists of the standard numerals, symbols, a syntax, and various algorithms for rearranging elements of the notational system. In computer programming, the notational system is the programming language itself. Finally, other disciplines have their own notational systems like free-body diagrams in mechanics, electric circuit schematics, chemical bond diagrams, etc.

While the representation is not the domain, per se, effective use and deep understanding of the notation is typical of expertise. The representations support algorithms where basic operations can be performed with little or no understanding of the underlying concepts. While an expert does understand those concepts, and uses his or her understanding to make the notations, representations, and algorithms less brittle in their application, a novice often does not understand the concepts. That is, novices try to apply the notations, representations, and algorithms without reference to the concepts, making errors in the process which they have no way to notice. This notion of the different forms of expertise combining effectively is developed in more detail below.

3. Rules for referring to situations in the world. These rules allow the expert to apply concepts and representations to actual situations in the world. These rules allow the expert to extract neat, well defined tasks from the complexity and fuzziness of the real-world. The rules allow an expert to interpret a situation in the world in terms of the abstractions, represent that situation with standard representations, and make predictions or solve problems.

So, for example, experts at basic arithmetic know how to apply arithmetic representations and concepts to the task of determining the most economical size laundry detergent at the supermarket. Similarly, an expert in basic electricity recognizes the circuits created in the way an automobile battery's negative pole is attached to the frame while the positive terminal is attached through switches and wires to the car's various electrical devices that are also attached to the frame. Finally, a programming expert is able to use standard loop constructs to represent all the different kinds of repetition found in the world.

Part of what makes an expert is his or her facility with all three aspects of expertise. When beginning a task, rules of reference allow the expert to interpret the situation at hand in terms of the concepts. These concepts are represented with a representation which then can be freely manipulated. The representation gives considerable cognitive efficiency because it operates without the constraints that would be induced by a constant, detailed attention to mapping back to the actual world situation. When critical, however, the expert is able to constrain and guide the purely representational operations with theoretical principles from the concepts and practical limitations imposed by the situation in the world.

Building Expertise Into an ITS

How exactly do these three kinds of expertise map on to the proposed interface orientation for ITS? The interface needs to embody all three aspects - that is, provide the integration that is natural in an expert's performance. An interface should provide some concrete way to visualize a concept, relate directly to standard representations, and refer to situations in the world. For example, White and Horwitz [16] presents a series of microworlds designed to teach successively richer models of mechanics. Each model includes a physical representation of concepts like velocity and acceleration based on the movements of a ball on the computer screen. The physical representation is also designed to suggest certain standard notational approaches. Similarly, Anderson's geometry tutor [1], while providing a set of tools for managing proof subgoals and available theorems - conceptual elements, also uses standard geometry notation to label those subgoals. Finally in Bridge [3], our own intelligent tutor for teaching beginning programming (discussed in detail below), each programming concept has a distinctively shaped icon and syntax rules suggested by how different icons can fit together.

How Can An Interface Help Learners Acquire That Expertise

Giving an ITS an expert interface orientation places the student in the same problem solving context as the expert. This occurs in a series of ways:

1. The interface can off-load computations that a learner might need to do. In Bridge, for example, much of the detailed semantics of the programming constructs is handled for the student. Similarly, students using the geometry tutor do not need to do the detailed symbol manipulation required to apply theorems. In some ways this is cheating, the student is shielded from the full complexity of the task. In allowing the student to focus on the salient tasks and ignore the minor details, it can be enormously useful.
2. The interface can highlight the most salient tasks or features. In the real world, for example, the broken part rarely changes color and starts to blink. There is no way isolate key elements or subsystems if those elements or subsystems are physically intertwined. The interface-based ITS however, is free to do all these things.

BRIDGE: AN INTELLIGENT TUTOR FOR TEACHING PROGRAMMING

Research into how novices learn programming reveals that understanding the semantics of standard programming languages is not the main difficulty of novice programmers. Instead, success with programming seems to be tied to a novice's ability to recognize general goals in the description of a task, and to translate those goals into actual program code (see, for example, [4,9,13].) In Bridge we built a programming environment that supports a novice in working with plans that describe the goals and subgoals typical of programming tasks. By using plans that describe programming goals, Bridge allows for initial novice conceptions of a problem solution that are informal and sketchy. The Bridge environment features an iconic plan programming language with an editor facilities to control execution and support debugging. A complete discussion of Bridge can be found in [3].

Bridge supports a novice in the initial informal statement of a problem solution, subsequent refinement of that solution, and final implementation of the solution as programming language code. This is accomplished in three phases, discussed in detail in the rest of this section. To illustrate Bridge use, we discuss a student working on the Ending Value Averaging problem:

Write a program which repeatedly reads in integers until it reads in the integer 99999. After seeing 99999, it should print out the CORRECT AVERAGE without counting the final 99999.

Bridge Phase I: Informal Natural Language Plans

The first phase of Bridge involves an informal statement and refinement of the goals for the code. Empirical evidence [2,8] suggests that novice programmers bring a vocabulary of programming-like plans from everyday experience with procedural specifications of activities expressed in natural language. These plans come from experience with step-by-step instructions like "check all the student scores and give me an average" or "see that hallway, if any doors are open close them." These informal plans, however, are often extremely difficult for novices to reconcile with the much more formal plans used in standard programming languages. Note, for example, that both example phrases involve an iteration without any specific mention of a repeated action.

In phase I we provide a plan language based on simple natural language phrases typically used when people write step by step instructions for other people. For example, a student can construct the phrase "... and so on ... until 99999 is seen." Figure 1 shows an example with several such phrases. Such phrases represent the highest level at which a student can express intentions to the system. Because of the ambiguity in such phrasings, Bridge must understand the student's intentions based on several possible naive models of programming. For example, a common naive model of looping allows a student to construct a loop with a description of the first iteration followed by the phrase "and so on." Based on the particular phrasings constructed by the student, Bridge infers a particular naive model.

English step-by-step Solution

Read in . . . each integer

Print . . . each integer

. . . And So On . . .

Until 99999 is seen

Figure 1. Phrases from a phase I Bridge solution.

Bridge supports three different naive models of programming loops. Based on our empirical work [2], these models subsume most of the misconceptions novices have about simple programming loops. Although, there are many different English phrases used by novices, a simple pattern-matching approach to parsing the phrases is sufficient for distinguishing among the three models for each of the different plans.

The first phase of Bridge is an implementation of the kinds of operations and mental models normally expressed in English language step-by-step specifications written by non-programmers. These operations are vague and include significant implicit knowledge about the objects being operated on. Only a limited set of links are possible between the plans: ordering and nesting. Data communication links between the plans are implicit, reflecting the structure of natural language specifications.

Bridge Phase II: Iconic Programming Plans

In the second phase of Bridge a programming student refines the informal description of phase I into a series of semi-formal iconic programming plans. Figure 2 shows the Bridge screen as the student is working in phase II. In this phase the plans are schema-like structures which describe how goals are transformed into actual programming code (see [13] for a detailed discussion of these plans).

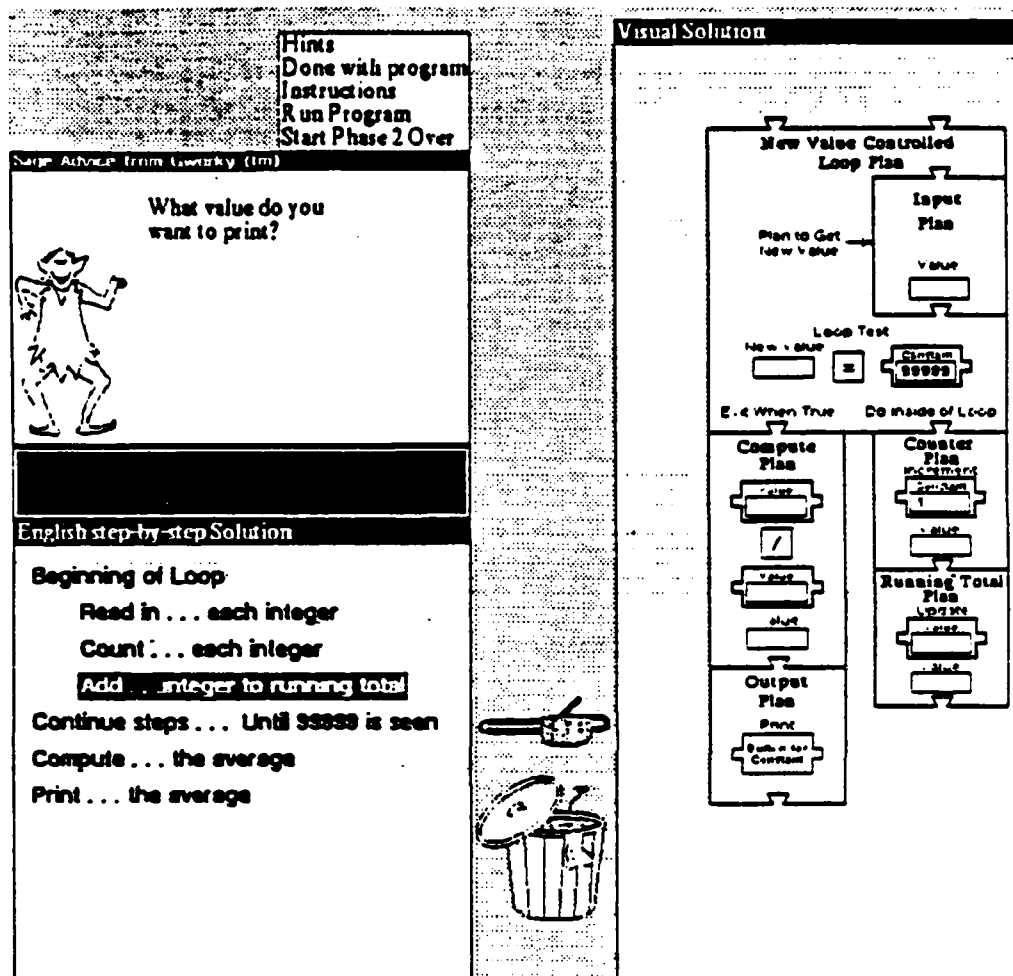


Figure 2. The Bridge system while a student is constructing a phase II solution, based on a complete phase I solution.

Plans have various elements that interrelate with the elements of other plans. So, for example, a counter plan has *initialize*, *increment*, and *use* elements, each with a particular relationship to the loop containing the counter. This interaction of elements often results in the plan implementation in standard programming constructs being dispersed across the program. A running total, for example, is implemented in Pascal with four statements, dispersed throughout a program: a variable declaration, an initialization above a loop, an update inside that loop, and a use below the loop. Spohrer and Soloway [14] have shown that plan to code translation errors account for many student errors.

In the second phase of Bridge students focus on relating various plan elements, but without compromising the fundamental plan structure and introducing the syntactic complexity required by standard programming code. Figure 3 shows a typical phase II solution to the Ending Value Averaging problem. Each plan is represented by a single icon. There are two kinds of links between the plan elements. Control flow links are expressed by attaching puzzle tabs to puzzle slots. Data flow

links are expressed by moving the small tiles with "ears" from the data source to the data destination. This is the way, for example, that the value of the Counter plan is associated with the average computation in the Compute plan.¹

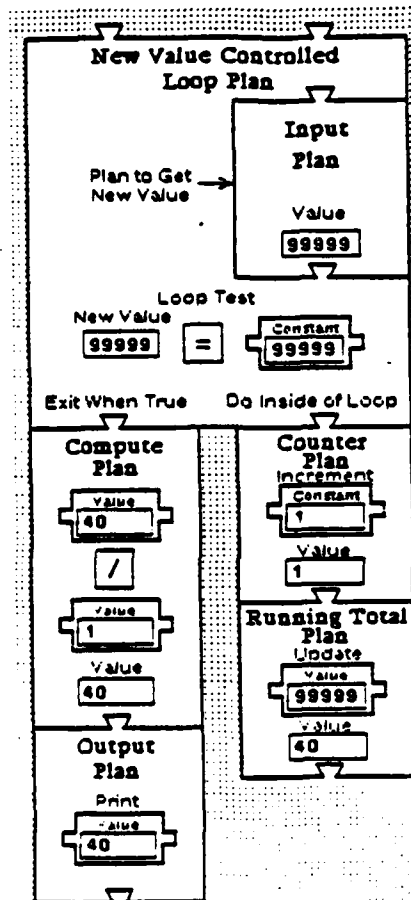


Figure 3. A typical phase II solution in Bridge.

Where phase I of Bridge represents the highest level plans, closest to the understanding of a non-programmer, phase II represents a lower level of plan. As the student works in phase II, links back to phase I are always available. Also, note that the phase II plans are executable and can be run by students using the Bridge tutor.

¹ A problematic design issue arises with the data value tiles. We would like to show the data flow graphically, but found the screen too cluttered when a wire was drawn from each data source to each data destination. Although we experimented with approaches that showed these links only when requested by the student, graphical data links were ultimately abandoned in favor of overall simplicity.

Bridge Phase III: Pascal Code

The third phase requires the Bridge student to translate the plan-based description of phase two into actual Pascal code. Students are provided with a Pascal structure editor (much like that of [5]), and an interpreter with a stepping mode. In this phase the user drops from the world of plans into a standard programming language.

Diagnosis with Plans

Any intelligent interface needs to infer user intentions from user behavior. In particular, the system must infer all mental activity from the actual actions performed by the user. In tutoring programming, for example, a standard intelligent tutor must reconstruct the student's entire mental activity between seeing a program specification and actually entering code in the machine. Such a reconstruction must account for both the correct and incorrect knowledge used by the student during design and implementation.

A tutor's reconstruction of a student's program is based on at least two kinds of knowledge. First, there must be a way for students to break the high-level goals of the problem statement into lower level goals. Second, there must be knowledge about how to translate low level goals into program code. Within the tutor we can represent these two kinds of knowledge as operators that transform one kind of structure into another. In addition to correct versions of these operators, the tutor must contain buggy versions representing common student misconceptions.

Using the knowledge base of correct and buggy operators a tutor can, in principle, use search techniques to reconstruct plausible accounts for errorful and ambiguous student specifications. This approach has been powerfully demonstrated in the programming tutor PROUST [6]. While the approach works, it is very costly in terms of both search time and knowledge engineering. The accomplishments of PROUST must be weighed against the large cost in knowledge engineering time - several hundred hours per problem tutored [7]. This knowledge engineering is particularly cost ineffective because the student sees so little of the results. That is, almost all of the knowledge engineering that has gone into capturing operators that describe how to apply programming knowledge is never seen by the student. Inside of PROUST, these operators are optimized for the the search task. They are not available in a form that could be presented to students, or used to assist students as they work towards a solution.

Bridge uses a different approach to reconstructing the student's intentions. Instead of attempting to reconstruct a student's entire reasoning from problem statement to final code in one step, Bridge has the student prepare intermediate solutions in plan languages that correspond to particular levels in the process of moving from problem specification to goals to code. This alleviates many of the difficulties of the PROUST approach. The search is more manageable because it has been broken up into a series of much smaller searches. In each of the smaller searches there are fewer relevant operators to try and less reasoning "distance" to span between the user's surface behavior and the solution the tutor is trying to reconstruct.

In addition, the Bridge approach simplifies the knowledge engineering. The fewer operators in each search correspond to a smaller overall catalog of operators to be specified. In addition, the smaller search spaces make it easier to tell when the space of possible correct and errorful versions has been covered.

A FRAMEWORK FOR UNDERSTANDING AND DESIGNING INTERFACES-BASED ITS

In previous sections I have discussed the value of an interface-based approach to ITS and the Bridge system as an example of this approach. This section presents a general architecture to support interface-based ITS. The architecture has the following two features:

1. Capturing an analysis of the educational process needed to move from novice conceptions of a task to expert (that is, rich) conceptions of a task so that it is explicitly available to the learner.
2. A visual interface and corresponding knowledge structuring approach that simplify and organize the elements captured in step 1.

Plans: Organizing An Interface-Based ITS

We propose a computer system/human-computer interface based on a simple set of icons that direct the machine. Each icon represents an operation and/or object that is important to the user at a particular level of detail. We call these operation/object elements *plans*. Plans are intended to capture common experience and standard operations. A plan set provides a set of tools at a particular level of detail. In addition to the plans, there are a few simple links that allow plans to be tied together in various ways (see figure 4).

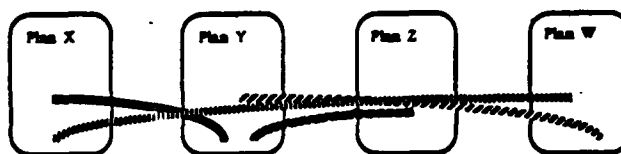


Figure 4. Plans used to create a simple computer application.

So far, we have a simple computer application that presents itself as a few modular components. Users are allowed to tailor these components in few limited ways to do their task. Our notion (and figure 4) should be viewed as a high level caricature - the details of the user interface probably will specialize the links and visual appearance in some way particular to the application. For example, we can imagine a simple spreadsheet where the links are connections between cells implicit in the spreadsheet formulas. The key idea is visual constituents to represent the key system elements, and a simple set of underlying links that support the basic interactions between those elements.

How is such a system expanded to include experts? We propose that each of the plans can be "opened up" revealing a whole new set of plans, linked together in such a way that they describe the implementation of the top level plan. That is, the plans at the top level actually are a high level summary of some collection and organization of lower-level plans. This plan hierarchy goes on for many levels (see figure 5). Moving down a level in the plan hierarchy is viewed with two perspectives:

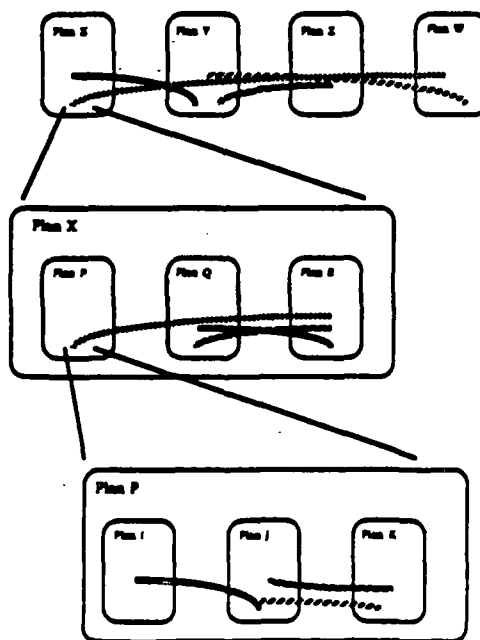


Figure 5. Interface based on a full plan hierarchy.

1. When you open a plan up to show the lower level set of plans that implement the original plan, you are requesting an explanation of how the top level plan works.

2. Simultaneously, you are stating your willingness to confront the additional data, additional complexity, or deeper abstraction inherent in the lower-level set of plans.

Crucial to this approach is that traveling down a level corresponds to exactly one new idea that must be understood by the user. Designing of the plan hierarchy should look more like a well designed course than a classic "top-down design" from the structured programming literature. Good teachers build elegant scaffolds through the complexity of their subject matter domain. The student is led with care and attention to avoiding the knotty complexities before the student actually needs to confront those complexities. Good teachers carefully select examples that are complete and accurate, as far as they go, but leave out important details that are irrelevant for that example. That is, the instructor designs the curriculum so the student only sees problems that are the right level of challenge for the student's current understanding. So, for example, elementary school students learning division begin with pairs of numbers that yield whole number results, i.e. $36 \div 6$, leaving remainders for later lessons.

Questions For Building A Plan Hierarchy

As far as it goes, the above proposal is interesting, but not at all complete. There are many key questions to be answered before it is of any practical use. These questions are as follows:

1. How do you cut up the task domain? Perhaps the most important questions - how do I cut the task domain into manageable pieces? How does one factor the many concepts and abstractions that make up any interesting application domain into the various plans and links?
2. What do you put at each level of the hierarchy? For a given application, what should be the highest level plans and what should be put in each successive layer? Is there one consistent layering, or do different task foci cause radical changes to the decisions about the layers to be presented?
3. What is the semantics of the plan icons? Exactly what semantics should be supplied to the plans? What of their internal structure is visible before the plan is "opened up". What kinds of values can be stored in plans and how are those values represented? What kinds of computations can be specified in plans and what does the specification look like?
4. What are the semantics of the links between plans? What do the connections mean? How many kinds of connections should there be? Should there be data flow links, control flow links, or both?

The answers to questions 3 and 4 represent the language and interface overhead; infrastructure required that makes no contribution to solving the application task. That is, these elements must be learned by a non-expert computer users without that learning directly contributing to solving any task of interest to the end user. This argues for making as simple decisions as possible when answering the questions about semantics that are presented above. The final section of this paper proposes an answer to questions 3 and 4.

Note that object-oriented programming community has confronted these questions in its emphasis on the idea of "reusable software modules" that can be plugged into any application. In practice, this is possible a lot less often than in theory, particularly for modules taken from more developed applications. That is, reusability gets less and less likely as you get closer and closer to providing real end-user functionality (as opposed to extending the toolkit available to programming experts).

We think there is a crucial missing component from the object-oriented programming approach: decisions about module composition are made on technical grounds; i.e. "what is the most powerful abstraction", or "what is most flexible combination of parameters". We are proposing to make decisions about module contents on pedagogical grounds; i.e. "what's the simplest useful idea that can be packaged into a module" and "what's the single abstraction that provides the next level of functionality for the user". While the lower-levels of a plan based system look like what might be designed by a systems programmer, the upper-levels are more like the work done by a knowledge engineer designing an expert system or like a teacher designing a course plan².

²Note, these two notions are probably not all that different.

MATISSE: A PLAN LANGUAGE FOR EDUCATIONAL MICROWORLDS

Based on the general discussion of a plan hierarchy, I propose a new language, Matisse³ to simplify the design and development of educational microworlds. In particular, Matisse is designed to support the design of actual plans that can be manipulated on the screen. The language is designed with the following goals:

- It should allow a microworld designer to base his or her design on an explicit mapping between an abstract domain and the graphical domain supported by the microworld.
- It should naturally support what is currently known about the effective design of microworlds.
- It should be usable by educators. These users need not be programmers. The language will hide the programming complexities inherent in direct manipulation interfaces.
- It should support the design, implementation, and delivery (in a classroom) of educational microworlds. The systems designed with Matisse must be efficient enough to work with students without reimplementing in standard languages (e.g. C, BASIC, etc.).

Example Use of Matisse

To illustrate Matisse I present a Matisse design for the strips and tiles microworld for teaching rational numbers in elementary school. In a "strips and tiles" representation of fractions unit rectangles are divided into equal sized sections (see figure 6). The student controls the number of section divisions and can color individual sections. The idea is that the colored section represents a fraction's numerator while the total number of sections represent the denominator. (see [10] for more details on this microworld). The Matisse system is not implemented, so the following description should be viewed as an illustration of the style to be used in developing a Matisse microworld.

³The name honors the paper cut-outs produced by Matisse. These works are made up of simple, iconic shapes cut out of construction paper. A goal for our microworlds is that they realize the dynamics and vibrancy in Matisse's compositions.

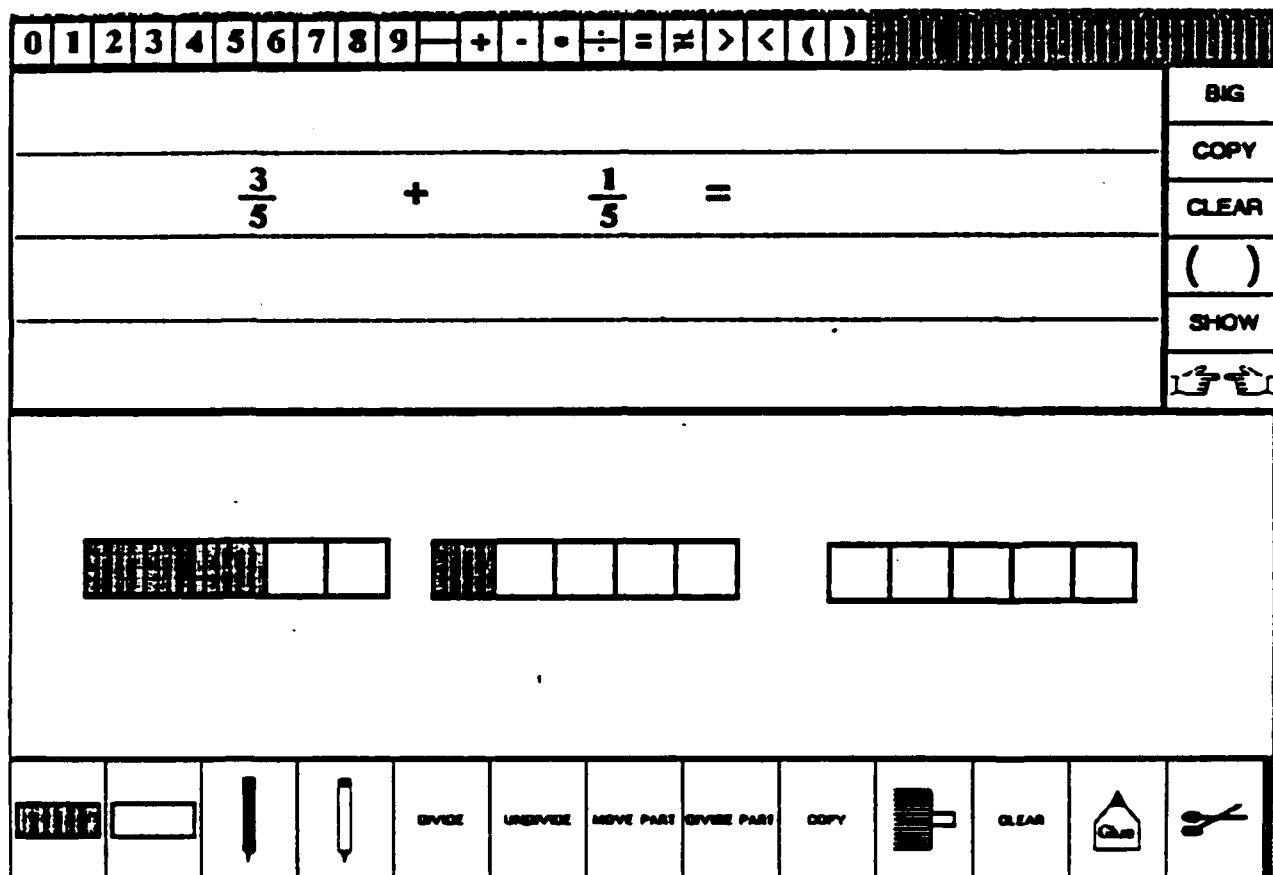


Figure 6. Example screen from the strips and tiles microworld.

That strips and tiles world has the following basic screen objects:

- A window to contain the strips and tiles microworld.
- Blank unit strips that can be subdivided into equal sized sections, including:
 - Sets of lines that evenly divide a unit strip into some number of equal sized sections
 - Shaded sections, corresponding to one of the sections in the current set of dividers. Each section may or may not be shaded. All shaded divisions may or may not be pushed together at one end of the unit strip.
- Pen used to indicate shading of a section
- Eraser used to indicate unshading of a section
- Stockpile of blank unit strips
- Dividing tool use to indicate divisions
- Fractions in standard notation
- A notebook window showing a fraction in standard notation. This window is the same size and shape as the strips and tiles window. Fractions in this window show are placed in the same relative position as the strips whose values are represented.

A Matisse discription of this microworld consists of a discriptions of each of those objects. Actually, a prototypic version of each object is described. All microworld elements are implemented as an instance of a prototypic component. A prototype indicates the screen appearance, properties, and relationship to instances of other prototypes. All instances of a particular prototype have their own values for internal properties while sharing the behavior specified in the prototype. So, for

example, all unit strips share the operation that allow their division, but each instance of a unit strip can have a different number of divisions.

Continuing with the example, the unit strip has the following properties:

- *Picture* - In this case the picture is selected from a library of building blocks. One of the standard building blocks is a rectangle. The microworld designer can adjust the height and width of the rectangle, and the thickness of the outer line.
- *Division Kind* - A rectangle can be divided into sections with horizontal or vertical divisions. For the strips and tiles microworld the *Division Kind* property specifies vertical divisions. Other kinds of pictures have other kinds of divisions. For example, Circles can have pie shaped divisions as well as horizontal and vertical divisions. All closed figures can be divided with a grid.
- *Division Count* - This property of the rectangle specifies how many divisions there should be.
- *Value* - The unit rectangles have a value that is a function of the number of divisions and how many of those divisions are shaded. The value can be expressed either of two different ways, depending on the desires of the microworld designer:
 - An ordered pair consisting of the number of divisions and a count of the shaded divisions.
 - A rational number created by dividing the number of divisions by the number of shaded divisions.
- *Position* - The location of the unit strip on the containing window.

Each of the unit strip properties can be specified with a *library element* - some predefined object available in the Matisse library, a *value* - a number, text string, or geometric figure - or as a *constraint equation*. Constraint equations allow the microworld designer to express relationships that are to be enforced whatever other behavior occurs in the microworld. To continue the example, the properties of the unit strips have the following values:

- *Picture* - library: rectangle (1in,3in)

This specifies a 1in by 3in rectangle to be used as the picture for the unit strip.

- *Division Kind* - library: vertical divisions, shadable

This tells Matisse to allow vertical divisions in the rectangle, and provide for the shading of those divisions.

- *Division Count* - value: 0

This is an initial value. This field can (and will be) changed by user actions (described below).

- *Value* - constraint: Ordered Pair(Shaded Division Count, Division Count)

This constraint equation says that the value of the unit strip is an ordered pair made up of the count of the divisions that are shaded and the number of divisions. Actually, no constraints are introduced at this point, except that this value, and any values that depend on this value, are updated automatically whenever the Shaded Division Count and the Division Count change.

- *Position* - value: Screen Position (StartingX, StartingY)

This is an initial value for the screen position. StartingX and StartingY can be replaced by some particular values. Note that the operator Screen Position is really a kind of ordered pair, specialized for representing screen position.

Now that the properties are specified, we must specify the behaviors for those properties that have a behavior:

- *Division Count*

always Division Count is ≤ 12

This behavior enforces a constraint that the division count must be less than or equal to twelve. This is required because the screen cannot resolve anything smaller.

whenever Left Button of Mouse is Down
 and Type of Cursor is Division Tool
 and Location of Cursor is Strip

then Ask User "How many divisions?"
 and Division Count becomes User's answer

This Matisse behavior specifies that the user can set the division count by picking up the Division Tool (how that is done is specified as part of the behavior for the Division Tool), and clicking it over the unit strip.

- *Position*

always Position of Strip inside Microworld Window

This Matisse behavior says that the strip must stay inside the microworld window.

That is the entire specification for the unit strips. A number of other objects need to be specified as well, but to simplify this exposition, only a one of those descriptions will be shown here.

- *Pen*

whenever Left Button of Mouse is Down
 and Type of Cursor is Normal
 and Location of Cursor is on Pen Icon

then Cursor becomes Pen Cursor

whenever Left Button of Mouse is Down
 and Type of Cursor is Pen Icon
 and Location of Cursor is Division of Strip

then Shade Division of Strip

CONCLUDING REMARKS

Interface-based architectures for ITS provide opportunities for delivering both more powerful tutors and more effective knowledge engineering. This paper has discussed three systems: a working tutor, a general architecture, and language for implementing parts of that architecture. These systems should not be seen as an attempt at systematic coverage but rather as provocative data points.

ACKNOWLEDGEMENTS

Work reported here was supported by the Office of Naval Research under contract numbers N00014-83-6-0148 and N00014-83-K0655 and by the the Air Force Human Resources Laboratory under contract number F41689-84-D-0002, Order 0004. Any opinions, findings, conclusions, or recommendations expressed in this report are those of the authors, and do not necessarily reflect the views of the U.S. Government.

Stellan Ohlsson and Stewart Nickolas designed the strips and tiles microworld. Robert Cunningham implemented Bridge and designed many of its features.

LIST OF REFERENCES

- [1] Anderson, J.R., Boyle, C.F., & Yost, G. (1984) The Geometry Tutor. In A. Joshi (Ed.), *Proceedings of the Ninth International Joint Conference on Artificial Intelligence* (pp. 1-7). Morgan Kaufmann Los Altos, CA.
- [2] Bonar, J., & Soloway, E. (1985) Pre-programming knowledge: A major source of misconceptions in novice programmers. *Human-Computer Interaction*, 1, 133-161.
- [3] Bonar, J., Cunningham, R., Beatty, P., & Weil, W. (1988) Bridge: Intelligent Tutoring With Intermediate Representations. Technical Report, Learning Research and Development Center, University of Pittsburgh, Pittsburgh, PA 15260
- [4] Eisenstadt, M., Laubsch, J., and Kahney, H. (1981) Creating Pleasant Programming Environments for Cognitive Science Students. *Proceedings of the Third Annual Cognitive Science Conference*. Berkeley, CA.
- [5] Garian, D.B., Miller, P.L. (1984) GNOME: An Introductory Programming Environment Based on a Family of Structure Editors. In *Proceedings of the Software Engineering Symposium on Practical Software Development Environments*, Association for Computing Machinery.
- [6] Johnson, L. (1986) *Intention-Based Diagnosis of Novice Programming Errors*. Morgan Kaufman, Palo Alto, CA.
- [7] Personal communication.
- [8] Kahney, H., & Eisenstadt, M. (1982) Programmers' mental models of their programming tasks: The interaction of real world knowledge and programming knowledge. *Proceedings of the Fourth Annual Conference of the Cognitive Science Society*.
- [9] Mayer, R.E. 1979. A Psychology of Learning BASIC. *Communications of the Association for Computing Machinery* 22(11).
- [10] Ohlsson, S., Nickolas, S.E., and Bee, N.V. Interactive Illustrations for Fractions: A Progress Report. Learning Research and Development Center, Knowledge and Understanding in Human Learning Technical Report KUL-87-03. December 1987.
- [11] Reiser, B., Anderson, J., & Farrell, R. (1985) Dynamic student modelling in an intelligent tutor for lisp programming. *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pp. 8-14.
- [12] Resnick, L.B., & Omanson, S.F. (1987) Learning to Understand Arithmetic. In R. Glaser (Ed.), *Advances In Instructional Psychology* (Vol. 3, pp. 41-95). Erlbaum, Hillsdale, NJ.
- [13] Soloway, E., & Ehrlich, K. (1984) Empirical Studies of Programming Knowledge. *IEEE Transactions of Software Engineering*, SE-10
- [14] Spohrer, J., Soloway, E., & Pope E. 1985. A Goal/Plan Analysis of Buggy Pascal Programs. *Human-Computer Interaction*, 1.
- [15] Towne, D.M. & Munro, A. (1988) The Intelligent Maintenance Training System. In Psotka, J., Massey, L.D., & Mutter, S.A. (Eds.) *Intelligent Tutoring Systems: Lessons Learned*. Erlbaum, Hillsdale, NJ.
- [16] White, B.Y. & Horwitz, P. (1988) ThinkerTools: Enabling Children to Understand Physical Laws. *Cognitive Science*.

Generating Explanations in an Intelligent Tutor Designed to Teach Fundamental Knowledge¹

Bruce W. Porter
Liane H. Acker
James C. Lester
Art Souther

Department of Computer Sciences
University of Texas at Austin
Austin, Texas 78712

Abstract

Providing coherent explanations of domain knowledge is essential for a fully functioning Intelligent Tutoring System (ITS). Current ITSs that generate explanations from the underlying representation provide a limited solution because they place restrictions on the form and extent of the underlying knowledge. However, generating explanations in tutors that are designed to teach the kind of foundational knowledge conveyed in most introductory college courses poses special problems. These problems arise because this knowledge is broad, complex, and contains multiple, highly-integrated views.

In this paper we propose an explanation generation system for the field of botany that provides a general solution. The system uses domain-independent knowledge in the form of *view types* to assure that the knowledge incorporated in the explanation is relevant and coherent. We show how view types are used in designing explanation strategies to answer the question "How is X defined?". We conclude by proposing extensions to the explanation generator that enable the processing of other question types and that take into account specific knowledge about the student and dialogue history.

¹Support for this research was provided by the Army Research Office under grant ARO-DAAG29-84-K-0060 and the National Science Foundation under grant IRI-8620052.

INTRODUCTION

Providing coherent explanations of domain knowledge is essential for a fully functioning Intelligent Tutoring System (ITS). There are two approaches to providing coherent explanations: presenting "canned text" and generating explanations from the underlying knowledge representation. Generating explanations offers several advantages. First, systems that generate explanations do not have to anticipate every question, thus they may be able to provide explanations even for unexpected questions. Second, the exact form of the explanation can be shaped to fit the current situation and student. Third, as the domain knowledge changes, canned explanations must be examined to determine if they are still consistent with that knowledge. In contrast, generated explanations are always consistent with the knowledge. Finally, generated explanations can be used by the tutor directly in tasks like diagnosis and evaluation. For example, they might be used to evaluate the quality of a student explanation or to diagnose the basis for a faulty conclusion.

Current ITSs provide a limited solution to explanation generation. Their success results either from natural or imposed limitations on the form and extent of domain knowledge. These limitations include dedicating the ITS to a single task [3,1,4], characterizing the domain knowledge by a relatively small number of rules or axioms [7,6,1], covering only a small portion of a domain [2], and keeping viewpoints separate in the knowledge base [1,7].

An important class of tutors requires a broader solution to generating explanations than these ITSs have provided. The domain of these tutors is the foundational knowledge conveyed in introductory college courses. For most subjects, this knowledge broadly surveys the domain, contains multiple, highly-integrated views, and is not reducible to a small number of principles or axioms.

These properties of foundational knowledge pose special problems for an explanation generator. It must access the relevant knowledge and generate adequate explanations from the multiple views that are typically represented. In addition, any ITS that generates explanations must have a way of controlling the degree of elaboration and the amount of detail provided so that the definitions are adequate but not overwhelming.

In this paper we present an *explanation generation system* for the domain of botany that addresses these problems. To assure the coherence and relevance of the explanations that are generated, this system employs *view types*, which are used to provide a viewpoint

on the knowledge within which to couch the explanation.

We begin by describing our extensive knowledge base in the domain of botany. The knowledge base contains foundational knowledge described with multiple, integrated views of botanical objects and processes. We then describe an explanation generation system that uses view types and default knowledge about the student. We conclude by proposing to customize these explanations using a student model and dialogue history.

REPRESENTING FOUNDATIONAL KNOWLEDGE

Botany is representative of domains with foundational knowledge. The botany knowledge base we are constructing currently contains over 4,000 concepts in the areas of anatomy, physiology, and development [5]. Nonetheless, this knowledge is only a small portion of the information contained in an introductory botany course. This knowledge is oriented around botanical processes (such as *growth* and *water absorption*) and the objects that participate in them (such as *root system*).

A comprehensive representation of each concept requires multiple views. For example, glucose may be viewed as a 6-carbon ring structure, a sugar, the product of photosynthesis, the raw material for fermentation and respiration, the building block for polysaccharides, the most essential nutrient provided by plants to animals, an important intermediate in many biosynthesis processes, an important osmotic particle, etc.

The "backbone" of the knowledge base is a collection of related botanical objects and processes (Figures 1 and 2). The relations support the inheritance of facts from general concepts to specific concepts. For example, photosynthesis is a kind of biological production event, whose raw materials are substances the plant must be able to assimilate. The output includes organic compounds which are more complicated than the raw materials and may include other organic and inorganic compounds in the form of by-products and waste-products.

Event Hierarchy

All relations are specialization

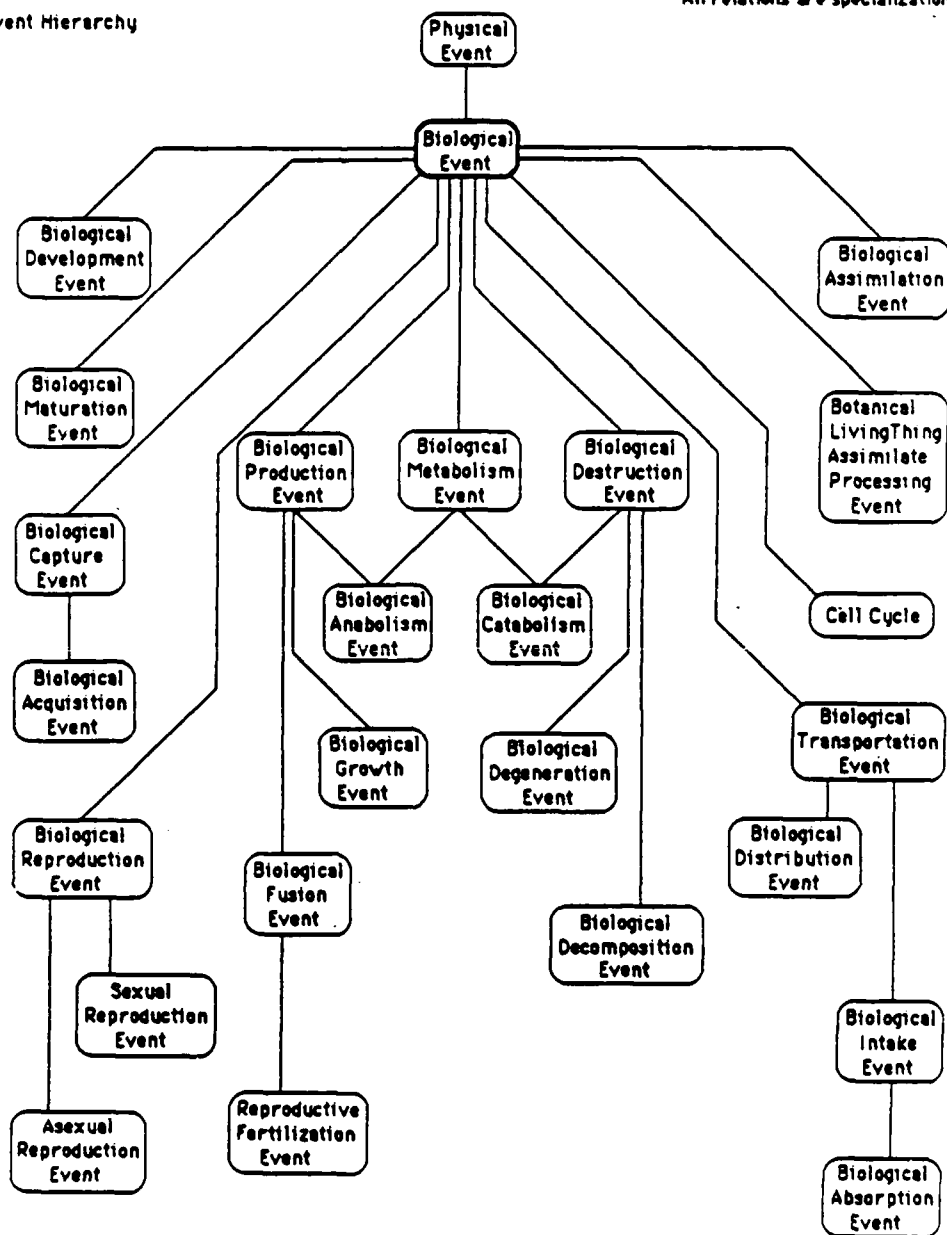


Figure 1

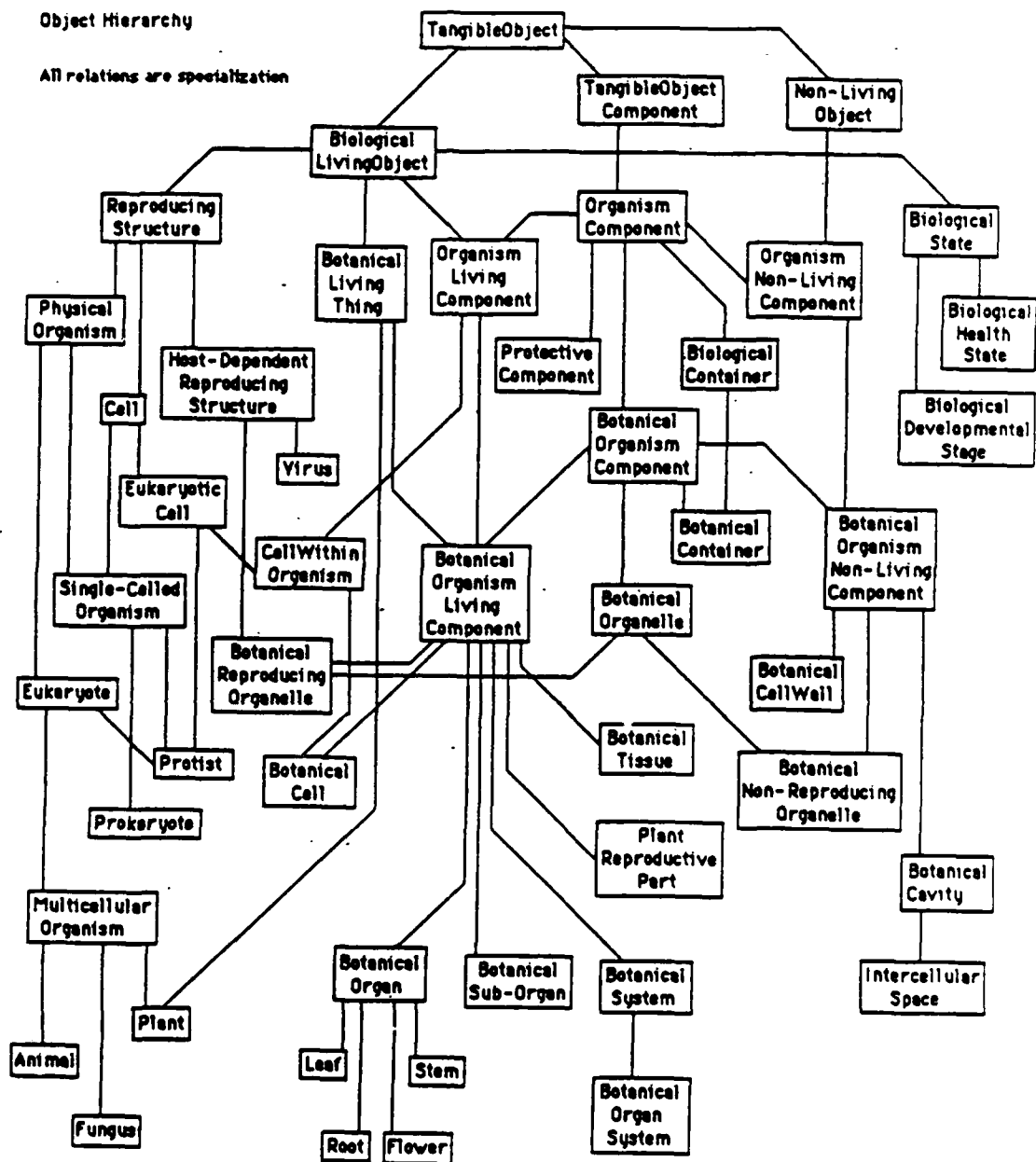


Figure 2

Building on this ontology, each concept is represented with a *frame*, which contains *slots* that record relations to other concepts. For example, Figure 3 describes the *fruit development* process, its relations to the objects it affects, and its subprocesses. The structure of objects and processes is represented with constraints among the parts: for example, the *grown seed* is constrained to be contained by the *developed pericarp*, and *fruit development* is constrained to occur before *fruit ripening*.

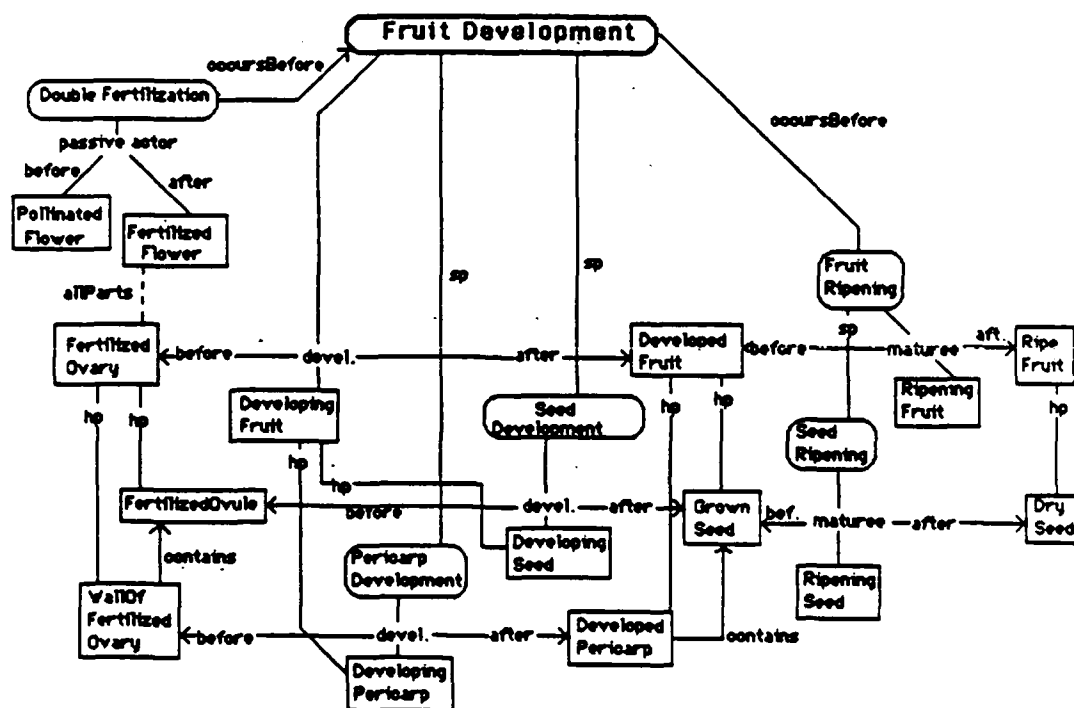


Figure 3

The foundational knowledge of botany, as in most fields, is broad and complex. This affords considerable flexibility in its use, but poses problems for access: how can an ITS locate and structure the knowledge relevant for answering a particular question?

AN EXPLANATION GENERATION SYSTEM: INITIAL RESULTS

When generating coherent explanations from a foundational knowledge base, it is necessary to constrain the information presented so that the explanations are coherent and contain only relevant knowledge. To accomplish this, we employ a small number of *view types*, which are used to define the context for the explanation. Strategies derived from the view types then generate the explanation within the contextual constraints defined by the view type.

Each explanation generation strategy is designed to answer a given class of questions the student might ask. In our examination of student questions, we have found that they fall into a hierarchy of question types. Each question type represents a class of equivalent questions, where two questions are considered to be equivalent if they can be put in the same form without losing meaning. This form is used to select the appropriate explanation generation strategy.

We will describe in detail the notion of view types and the question types they address. Then we will show how strategies derived from the view types can provide answers to the question type "How is X defined?".

View Types

Each coherent area of a knowledge base represents a particular viewpoint on the domain knowledge. For example, an explanation about pollen might use the viewpoint of "pollen as an actor in plant reproduction" or "pollen as an object composed of other parts and substances" or "pollen as a nasal irritant."

Although there are an unlimited number of *viewpoints* possible, we propose that a small number of *view types* are sufficient to characterize all viewpoints. The view types that we have developed are the *functional*, *modulatory*, *structural*, *class-dependent*, and *attributional* view types.

The functional view type considers the role of some object in some process. For example, the viewpoints

- Pollen as an actor in Plant Reproduction
- Chloroplast as the producer in Plant Photosynthesis

both employ the functional view type.

A functional viewpoint necessarily includes a slot that represents some kind of *actor in* relation, such as *producer*, *agent*, and *raw material*. Sometimes this slot relates the object and process of interest directly. Often there is only an indirect relationship. For example, a part or specialization of the object may be an actor in the process specified, rather than the object itself. Thus a functional viewpoint that relates the object and process includes an obligate *actor in* type slot and certain other "permissible" slots (e.g., *has part* and *specialization*) that provide intermediate links between the object and process. These slots are permissible because their presence does not invalidate the functional relationship between the object and process. For example, it can be said that one of the functions of the seed is to protect the plant embryo, though strictly speaking it is the seed coat, a part of the seed, that protects the embryo.

The modulatory view type considers how one object or process affects another. An example of a modulatory viewpoint is "Sunlight as an influence on Plant Growth" or "Embryo Growth as a cause of Seed Coat Rupture." A modulatory viewpoint necessarily includes at least one regulatory slot, such as *causes* or *inhibits*. Other slots also may be included, as with the functional view type.

The structural view type considers an object or process in terms of its substructures or superstructures. These structures may be either temporal or spatial.

For example,

- Photosynthesis as the Light Reactions followed by the Dark Reactions
- Seed as the structure consisting of a Seed Coat, Endosperm, and Embryo
- Flower as Flower Bud Stage followed by Opening Flower Stage followed by Mature Flower Stage

are all substructural viewpoints. Analogous superstructural viewpoints are

- Light Reactions as the subprocess of Photosynthesis preceding the Dark Reactions
- Seed Coat as the part of a Seed containing the Endosperm and Embryo
- Opening Flower Stage as the stage of Flower development following the Flower Bud Stage and followed by the Mature Flower Stage.

As illustrated by these examples, a structural viewpoint includes those constraints that specify how the temporal or spatial parts are interconnected.

The class-dependent view type considers a concept in terms of how it fits into a class hierarchy. The class-dependent view type has two subtypes: categorical view type and enumerative view type. The categorical view type considers a concept in terms of the properties and relations it inherits from one of its generalizations (superclasses). For example, "Flower as Reproductive Organ" and "Photosynthesis as Energy Transduction Process" are categorical viewpoints. The enumerative view type considers a class concept in terms of its instances (members) or specializations (subclasses). An example of the enumerative view type is "Plant Reproduction as Sexual Plant Reproduction or Asexual Plant Reproduction."

The simplest view type is the attributive view type. This view type considers a concept in terms of its properties or attributes. Property slots, such as color and weight, have values that fall along some range or spectrum. Unlike the other view types, the attributional view type involves properties rather than relations.

Since each of these view types concerns the relationship of the concept of interest to at most one other concept, it is straightforward to specify a particular viewpoint using these view types. All that is required is to specify the concept of interest, the view type, and the *reference concept*, that is, the concept to which the view type relates the concept of interest. For example, (Pollen Functional-View-Type Plant-Reproduction) specifies pollen from the viewpoint of its functional role in plant reproduction. Thus a view type is instantiated to a particular viewpoint by specifying the concept of interest and the reference concept.

An ITS can use view types to generate coherent explanations in the following way. The question asked by the student determines the concept of interest. The system then selects a view type using information in the student's question and meta-level knowledge about the applicability of the view types to the particular question type. The explanation generator has a strategy for each valid mapping between a question type and a view type. It uses the appropriate strategy to retrieve from the knowledge base the network of frames, slots, and constraints that forms the basis of a coherent answer to the student's question. The strategy uses the concept of interest and the reference concept to guide access to the knowledge base.

Question Types

To develop strategies for generating coherent explanations in response to student questions, an ITS must have a predefined set of question types. A question type is a template for a class of questions that have the same kind of conceptual representation and that are answered using the same explanation generation strategy. It is important to classify questions on the basis of conceptual representation rather than syntactic form because the same question can be phrased in many ways. However, one way to facilitate the process of classifying student questions is to restrict the allowable syntax for each question type so that the syntax of a student's question uniquely determines its meaning, and thus its question type.

We have developed a hierarchically structured set of question types. When the student asks a question of a particular question type that has subtypes, the stand-alone explanation generator uses a default knowledge structure to transform the question into a more specific question covered by one of the subtypes (or into a combination of more specific questions). The customized system will be able to use the student model, the dialogue history, and the knowledge base to make such transformations. In this way, the system can apply very specific and direct explanation generation strategies even to a very general question. The hierarchy of question types we have developed is summarized below.

- How is X defined?
 - What kind of thing is X?
 - What are some examples of X?
 - Describe the spatial/temporal structure of X.
 - What are the properties of X?
 - In what events is object X an actor?
 - What events/objects does X affect (and how)?
 - What events/objects affect X (and how)?
 - * What conditions are sufficient/necessary for X?
 - * What would be the effect on X if condition C were true/false?
 - * What processes/conditions/quantities influence quantity Q of X?

- Compare/contrast X with Y.
 - How is X different from the prototypical Y?
 - How are the values for characteristic C of X like/different from the values for characteristic C of Y?
 - How is the spatial/temporal structure of X like/different from the structure of Y?
 - How is the role of object X in process P like/different from the role of object X in P?
 - How is the effect of X on Z like/different from the effect of Y on Z?
- What are the values for characteristic C of X?
- Why does X have the value V for characteristic C?
- Why doesn't X have the value V for characteristic C?
- How is X related to Y?
- How does object X perform process P?
- What concept has the following properties and relations?
- What would be implied if condition C were true/false?

For each question type an explanation generation strategy must be developed. If the question type represents questions that can be answered using more than one view type, then a strategy that can be applied to the question type is needed for each view type. For the definition and comparison question types described above, we have developed strategies for each of the five view types. In the next few months we plan to implement these strategies in a prototype explanation generation system. The next section is a brief discussion of a few of these explanation generation strategies.

Definition Strategies

Much of our work on explanation generation strategies has focused on the definition question type. We have taken a liberal interpretation of definition in which "How is X defined?" is taken to mean "Give me some information about X that will help me to understand its significance." Each view type is used to guide the construction of a

particular explanation generation strategy. The explanation strategies for each of the five view types are discussed below.

The functional view type strategy for generating definitions is as follows:

- The system collects all values found on *actor in* slots on the frame representing the concept of interest.
- If any of these values is the reference concept, then the functional relationship has been established. If none of them are, the system determines if any of the values are specializations or subevents of the reference concept. (Here the reference concept is restricted to processes.)
- If the above fails, the system attempts to establish a functional relationship between one of the specializations or parts (either spatial or temporal) of the concept of interest and the reference concept.

The class-dependent view type has two subtypes: the categorical view type and the enumerative view type. The definition generation strategy for the categorical view type attempts to show how the concept of interest is a specialization of the reference concept. The reference concept is restricted in this case to a generalization of the concept of interest. First, all slots, values, and constraints that the concept of interest inherits from the reference concept are collected. Then the system retrieves all values local to the concept of interest that appear on slots inherited from the reference concept.

The enumerative view type involves collecting either all the instances of a concept representation in the knowledge base or all the specializations. If an unmanageable number are found, then the system will present only a subset of the examples to the student. Unless requested by the student, the system selects the enumerative view type only as part of a definition using multiple viewpoints or for the definition of a superordinate category. A superordinate category, such as furniture, is sufficiently polymorphic to warrant a definition by example.

The structural view type strategy for generating definitions is conceptually the same regardless of if a spatial structural viewpoint or a temporal structural viewpoint is requested. The same basic strategy is used to explain the parts of an object, the stages of an object, and the subprocesses of a process. The only difference in the strategies for these three is the actual slots employed.

To give a spatial substructural definition of an object, the system reports the values and constraints on all *part* slots on the frame representing the object. The definition also includes constraints on how these parts are interconnected. To give a spatial superstructural definition of an object, the system reports the values and constraints on all slots that indicate what the object is part of, connected to, or contained in. In addition, it reports all constraints indicating how the object is connected to its superstructures, including its "neighboring" parts and interconnections.

The definition generation strategy corresponding to the modulatory view type has several varieties, depending on the nature of the concept of interest and the reference concept. Since either can be an object or a process, there is a strategy for each combination. A general tactic used is to determine if some quantity, such as size or rate, of one concept affects some quantity of the other concept through a qualitative relation. To determine how a concept (either an object or a process) affects another process, the system first determines if the concept appears in any constraints or preconditions on the process, or on one of its subprocesses or specializations. Failing this, if the concept is an object, it determines if one of the specializations or parts of the object affects the process. If still no modulatory relationship is established, then it searches for an *actor* in relationship. If the concept is a process, it tries to establish a chain of slots such as *causes*, *enables*, *prevents*, *inhibits*, and *facilitates* from the first process to the second. The search for such a chain also can involve specializations, generalizations, subprocesses, and superprocesses of the two processes. To establish a modulatory relationship between a process and an object, the system determines if the process or one of its specializations or subprocesses has the reference object as an actor. If this fails, it determines if the event affects some specialization or part (temporal or spatial) of the reference object.

If no modulatory link can be established using the above strategies, then it may be possible to construct an indirect modulatory link. For example, although there may be no representation in the knowledge base of how the amount of sunlight in a plant's environment affects the amount of glucose in the plant, such a relationship can be established by first establishing the relationship between the amount of sunlight and the rate of photosynthesis, and the relationship between the rate of photosynthesis and the amount of glucose. The system will search for such an indirect relationship only as a last resort and with a time limit imposed.

The definition strategy corresponding to the attributional view type involves collecting all slots on the frame representing the concept of interest that are property or attribute slots, such as color and weight. Once these slots have been collected, the values and constraints found on the slots are retrieved. Like the enumerative view type, the system will never use the attributional view type as the sole basis of a definition. Rather it will use it in combination with other view types to provide added detail to the definition.

CONCLUSION

An important task of Intelligent Tutoring Systems is providing students with coherent explanations. There are two basic approaches to explanation generation: presenting "canned text" and generating explanations directly from the representation in the knowledge base. Unlike using "canned text," generating explanations directly from the representation allows for the flexible use of explanations in the tasks of presentation, diagnosis, and interpretation. We have presented five view types that can be used to design explanation generation strategies for answering student questions.

Questions that can be put in the same form without loss of meaning constitute a question type. When a student asks a question, the system uses the form of the question to select the appropriate view type for the question. It then uses the explanation generation strategy associated with the view type for that particular question type to generate a coherent explanation from the underlying representation.

We have developed strategies from each of the five view types for the definition and comparison question types. The definition strategies, either singly or in combination, were sufficient to generate each of the 50 definitions selected from a botany textbook. In the next few months we plan to develop strategies for the other question types and to implement these strategies in a prototype explanation generation system.

Future work will include embedding the system described here in a customized explanation generator. This customizing system will be sensitive to the particular needs of the current student by accessing information contained in a student model. This requires designing a student modelling system to build and maintain a model of the student's evolving knowledge of the domain. To generate explanations that maximize discourse coherency, the customizing explanation generator will also maintain and use a dialogue history. We

will then perform a comparative analysis of the noncustomizing and the customizing systems to empirically test the effectiveness of explanation customization, as well as to gauge the cost of enhancing a stand-alone explanation generator with a customizer, a student model, and a dialogue history.

References

- [1] Brown, J.S.; Burton, R.R.; and de Kleer, J. (1982) Pedagogical, natural language, and knowledge engineering techniques in SOPHIE I, II, and III. In Sleeman, D.H.; and Brown, J.S. (Eds.) *Intelligent Tutoring Systems*. Academic Press, London.
- [2] Brown, J.S.; Burton, R.R.; and Zdybel, F. (1973). A model-driven question-answering system for mixed-initiative computer-assisted instruction. *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 3, 248-257.
- [3] Clancey, W.J. (1987) *Knowledge-based Tutoring: The GUIDON Program*. Cambridge, Ma.: MIT Press.
- [4] Hollan, J.D.; Hutchins, E.L.; and Weitzman, L. (1984) STEAMER: an interactive inspectable simulation-based training system. *AI Magazine*, vol. 5, no. 2, 15-27.
- [5] Porter, B.W., Lester, J., Murray, K.S., Pittman, K., Souther, A., Acker, L., and Jones. T. AI Research in the Context of a Multifunctional Knowledge Base: The Botany Knowledge Base Project. AI Laboratory Technical Report AI88-88, 1988. University of Texas at Austin.
- [6] vanLehn, K.; and Brown, J.S. (1980). Planning Nets: a representation for formalizing analogies and semantic models of procedural skills. In Snow, R.; Frederico, P.; and Montague, W. (Eds.) *Aptitude, Learning, and Instruction: Cognitive Process Analyses*. Hillsdale, N.J.: Lawrence Erlbaum Associates.
- [7] White, B.Y.; and Frederiksen, J.R. (1987). Qualitative Models and Intelligent Learning Environments. In Lawler, R.; and Yazdani, M. (Eds.). *AI and Education*. New York: Ablex Publishing Co.

REPRESENTING, ACQUIRING, AND REASONING ABOUT TUTORING KNOWLEDGE¹

Beverly Woolf

Department of Computer and Information Science
University of Massachusetts
Amherst, Massachusetts 01003
csnet: bev@cs.umass.edu

Abstract

This paper describes the process of encoding tutoring knowledge in a knowledge-based tutor and breaks it down into its component parts. *Knowledge representation* is explained in terms of modelling domain knowledge, human thinking, learning processes, and tutoring strategies. A uniform language is proposed to store tutoring primitives, including lessons, topics and presentations. *Knowledge acquisition* is described as a methodology for identifying and encoding the expertise used by teachers to reason about tutoring. *Control knowledge* is explained in terms of the machine's ability to first select a topic or response for an individual student and then customize its discourse or dynamically modify its examples, questions, or descriptions.

The paper describes how tutoring can be understood in terms of the Artificial Intelligence paradigm of knowledge and control. It shows how to represent knowledge computationally and how to express it as strategies and rules. Architectures are proposed for developing new systems and tools described to facilitate the process. The tools are now used in a generic and consistent foundation which has enabled us to represent, acquire, and reason about tutoring knowledge across several domains and from within several sites. Our goal is to enhance this framework and ultimately to produce a system in which "just plain folk", including psychologists, instructional scientists, and domain experts, can work directly on the machine to modify and upgrade tutors without the need for knowledge engineers.

¹This work was supported in part by a grant from the National Science Foundation, Materials Development Research, No. 8751362. It was also supported in part by the Air Force Systems Command, Rome Air Development Center, Griffiss AFB, New York, 13441 and the Air Force Office of Scientific Research, Bolling AFB, DC 20332 under contract No. F30602-85-C-0008. This contract supports the Northeast Artificial Intelligence Consortium (NAIC). Partial support was also received from URI University Research Initiative Contract No. N00014-86-K-0764.

1 BUILDING A TUTORING SYSTEM

We have evolved a generic and consistent foundation for representing, acquiring, and reasoning about tutoring knowledge. The big payoff has been that we can apply the framework and evolving theory to several domains and have developed or designed systems that tutor about statics, thermodynamics, time management, statistics, genetics, algebra word problems, and explanations. In this paper we name that foundation, draw it, and describe it.

We are not invested in promoting a particular tutoring strategy, nor do we advocate a specific intelligent tutoring system design. Rather, we build tools that allow for a variety of system components, teaching styles, and intervention strategies to be combined into a single framework. For example, Socratic tutoring, incremental generalizations, and case-based reasoning are just a few of the teaching strategies that we have implemented in this framework. Ultimately, we expect the machine to reason about its own choice of intervention methods, to switch teaching strategies, and to use a variety of tactics and teaching approaches, while making decisions about the most efficacious method for managing one-on-one tutoring.

We are aided in our work by colleagues in three states who apply the tools we develop to new domains and new user groups.² For example, colleagues at San Francisco State University have sent us several carefully built physics simulations on top of which we placed the tutoring formalism described here.³ These colleagues help us evaluate the tutors. One professor at City College of San Francisco used the statics tutor (Section 2.1) in a classroom and noticed deficiencies in the machine discourse which she amended in her verbal discourse with the students. She added examples or explanations to bolster the tutor's response, to diagnose student preconceptions, or to clarify the system's response. She delivered a list of additional discourse moves to us to be incorporated into the next version of the tutor.

1.1 Cycle of Development in Artificial Intelligence Systems

Development of intelligent tutors, like development of any Artificial Intelligence system, requires several iterative cycles: computer scientists and instructional designers have to first collaborate on the design and development of a system, additional collaboration is required to test it with students, and then the original implementation has to be modified and refined based on information gained through testing. This cycle is repeated as time permits.

1.2 Representation and Control

Artificial Intelligence programs are built by first defining and encoding the knowledge to be used and then by building control structures which define the way an interpreter will tra-

²Participant institutions include San Francisco State University, San Francisco City College, Trinity College in Hartford, CT., and State University of New York at Plattsburgh, NY.

³San Francisco State University, the University of Massachusetts, and the University of Hawaii are members of the Exploring System Earth Consortium. ESE, a group of universities and industries working together to build intelligent science tutors. The consortium is supported by the Hewlett-Packard Corporation.

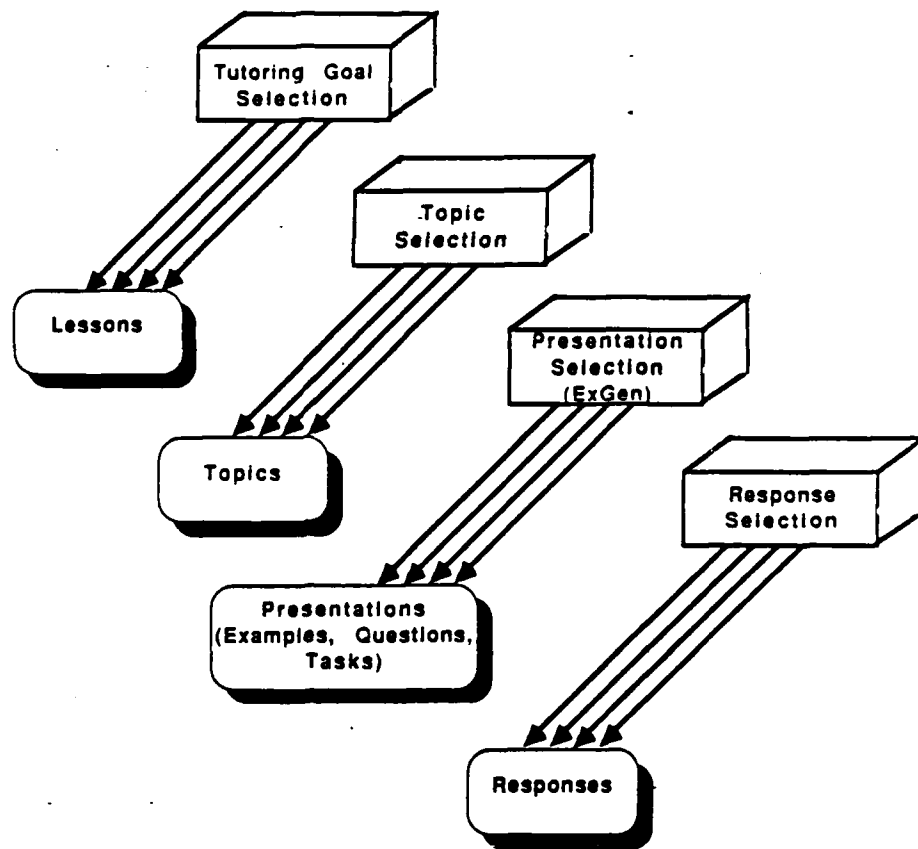


Figure 1: Representation and Control in a Tutoring System.

verse that knowledge. Knowledge representation refers to how knowledge is stored by a system to allow it to model the domain, human thinking, learning processes, and tutoring strategies. Knowledge bases store concepts, activities, relations between topics, and other quantities needed to make expert decisions. In particular, they store a variety of the lessons, topics, presentations, and response selections available to the tutor (see Figure 1).

Control refers to passage of an interpreter through those knowledge bases and its selection of appropriate pieces of knowledge for making a diagnosis, a prediction, or an evaluation. For tutoring, control structures are specified at the four levels indicated in Figure 1, separately defining control for selection of lesson, topic, presentation, and response selection.

Currently, our control structures are motivated by specific instructional and diagnostic goals: thus, for example, one control structure produces a predominantly Socratic interaction and another produces interactions based on presenting incrementally generalized versions of new concepts or examples. Control structures are specific to a particular level of control and are used separately to define the reasoning to be used for selecting a lesson, topic, presentation, or response.

Acquiring and encoding this large amount of knowledge, or the knowledge acquisition process, is difficult and time consuming. This paper describes our current efforts to perform knowledge acquisition and knowledge engineering and to represent and reason about this knowledge

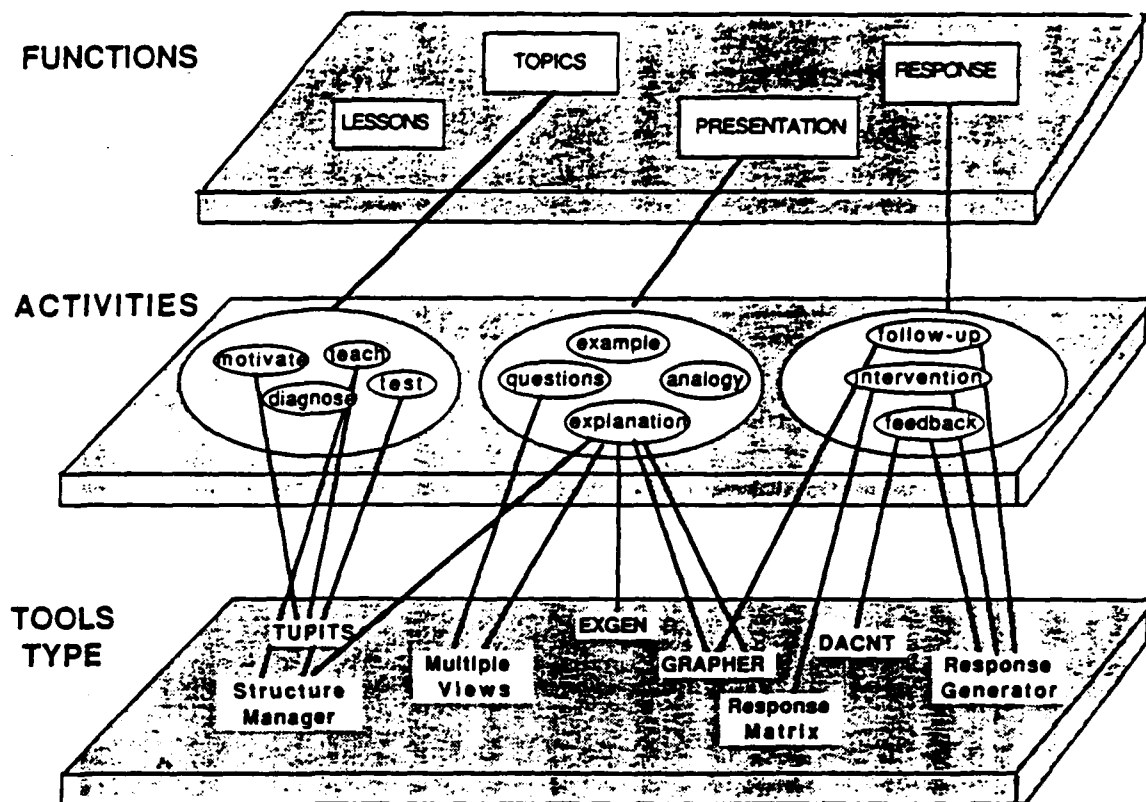


Figure 2: Tools for the Representation and Control of Tutoring Knowledge.

In particular, we have built a number of tools that facilitate the representation and reasoning about tutoring knowledge (see Figure 2). For each knowledge base (lessons, topics, presentation, or response), we consider the nature of the knowledge that must be accessed, such as the examples or questions (from the presentation knowledge base) or the activity the tutor must engage in, such as to motivate or teach a topic, or to provide follow-up. We have built tools to support each activity listed at the bottom of Figure 2. Only a few such tools will be described in this paper, namely TUPITS, Exgen, Response Matrix, DACTN, and multiple views.

We divide the discussion into two parts, separately presenting tools for representing tutoring primitives, or lessons, topics and presentations, and tools for representing discourse knowledge.

2 TOOLS FOR REPRESENTING TUTORING PRIMITIVES

We define tutoring primitives as those elements needed for tutoring, such as topics to be taught, specific tutoring responses, and possible student errors. Our knowledge bases hold a variety of examples, types of knowledge, tasks to be given to the student, and discourse states describing various human-machine interactions.

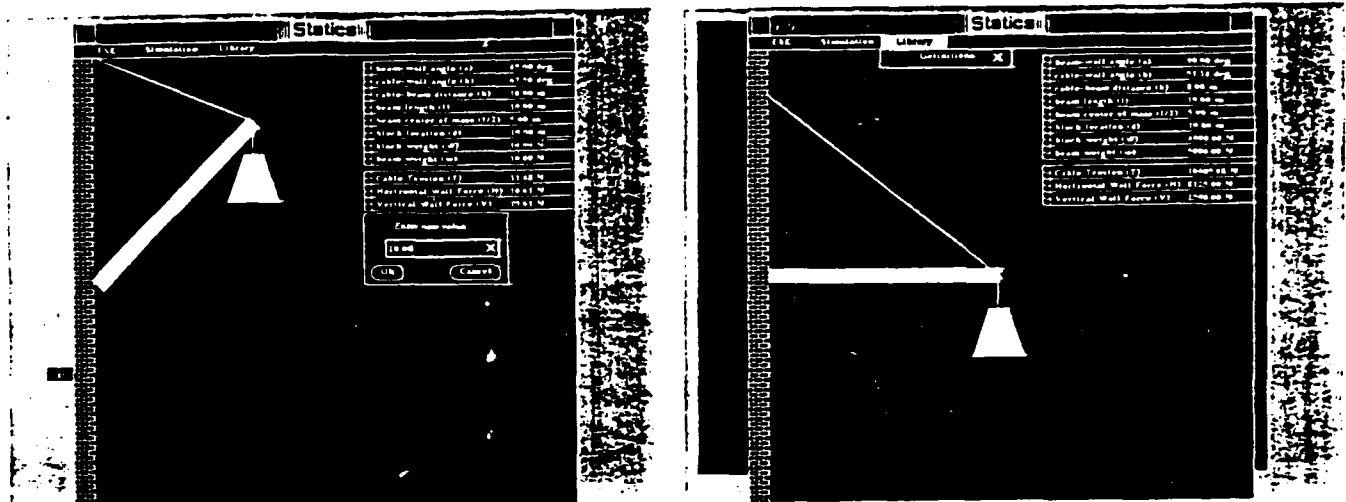


Figure 3: Statics Tutor.

2.1 Example Tutoring Primitives

As an example of how tutoring primitives are used, we describe two tutors we have built in conjunction with the Exploring Systems Earth (ESE) Consortium [3]. These tutors are based on interactive simulations that encourage students to work with "elements" of physics, such as mass, acceleration, and force. The goal is to help students generate hypotheses as necessary precursors to expanding their own intuitions. We want the simulations to encourage them to "listen to" their own scientific intuition and to make their own model of the physical world before an encoded tutor advises them about the accuracy of their choices. These tutors have been described elsewhere (see for example, [16]; [18]) and will only be summarized here.

Figure 3 shows a simulation for teaching concepts in introductory statics. In this example, students are asked to identify forces and torques on the crane boom, or horizontal bar, and to use rubber banding to draw appropriate force vectors directly on the screen. When the beam is in static equilibrium there will be no net force or torque on any part of it. Students are asked to solve both qualitative and quantitative word problems.

If a student were to specify incorrect forces either by omitting force lines or by including the wrong ones, the tutor makes a decision about how to respond. There are many possible responses depending on the tutorial strategy in effect. The tutor might present an explanation, a hint, provide another problem, or demonstrate that the student's analysis leads to a logical contradiction. Still another response would be to withhold explicit feedback concerning the quality of the student's answer, and to instead demonstrate the consequence of omitting the "missing" force: i.e., the end of the beam next to the wall would crash down. Such a response would show the student how her conceptions might be in conflict with the observable world and

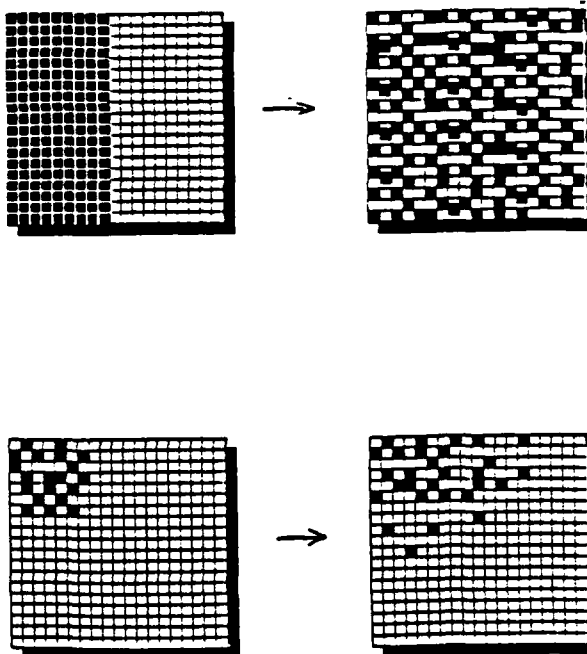


Figure 4: Thermodynamics Tutor.

to help her visualize both her internal conceptualization and the science theory.

A second tutor is designed to improve a student's intuition about concepts such as energy, energy density, entropy, and equilibrium in thermodynamics. It makes use of a very simplified but instructive simulated world consisting of a two-dimensional array of identical atoms (Figure 4 [1]). Like the statics tutor, the thermodynamics tutor monitors and advises students about their activities and provides examples, analogies, or explanations. In this simplified world the atoms have only one excited state; the excitation energy is transferred to neighboring atoms through random "collisions." Students can specify initial conditions, such as which atoms will be excited and which will remain in the ground state. They can observe the exchange of excitation energy between atoms, and can monitor, via graphs and meters, the flow of energy from one part of the system to another as the system moves toward equilibrium. In this way, several systems can be constructed, each with specific areas of excitation. For each system, regions can be defined and physical qualities, such as energy density or entropy, plotted as functions of time.

2.2 Representing and Reasoning about Tutoring Primitives

The topics, examples, explanations, and possible misconceptions about concepts to be taught in these two domains must be represented in the four knowledge bases described in Section 1.2. We use a network of Knowledge Units frames to explicitly express relationships between topics such as prerequisites, corequisites, and related misconceptions (Figure 5). An important notion about the network is that is declarative—it contains a structured space of concepts, but does not mandate any particular order for traversal of this space.

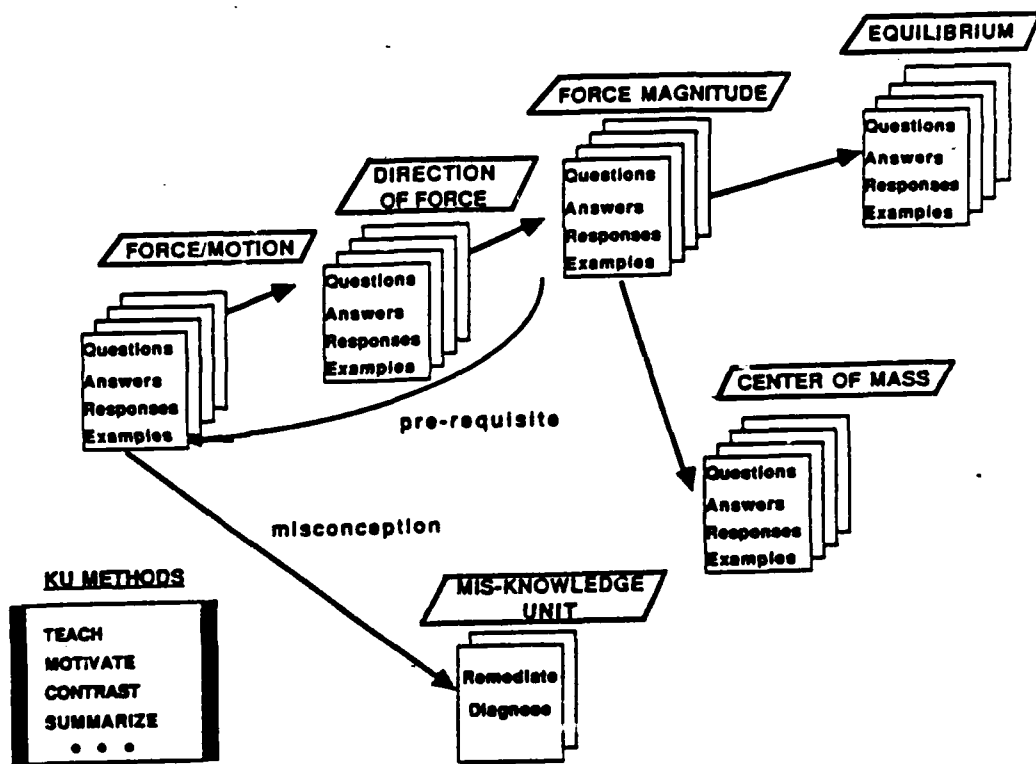


Figure 5: Hierarchy of Frames.

The network describes tutorial strategies in terms of a vocabulary of primitive discourse moves. It is implemented in a language called TUPITS⁴ which was used to build both the tutors described in Section 2.1. It is an object-oriented representation language that provides a framework for defining primitive components of a tutorial discourse interaction. These components are then used by the tutor to reason about its next action.

As shown in Figure 5, each object in TUPITS is represented as a frame and each frame is linked with other frames representing prerequisites, co-requisites, or triggered misconceptions. The primary objects in TUPITS are:

- Lessons which define high-level goals and constraints for each tutoring session (see [7]);
- Knowledge Units (KUs);
- MIS-KUs, which represent common misconceptions, wrong facts or procedures, and other types of "buggy" knowledge;
- Examples, which specify parameters that configure an example, diagram, or simulation to be presented to the student;
- Questions, which define tasks for the student and how the student's behavior during the task might be evaluated; and

⁴TUPITS (Tutorial discourse Primitives for Intelligent Tutoring Systems) runs on a Hewlett-Packard Bobcat and an Apple Macintosh II.

- Presentations, which bind an example and a question together.

MIS-KUs, or "Mis-Knowledge Units," represent common misconceptions or knowledge "bugs" and ways to remediate them. These are inserted opportunistically into the discourse. The tutoring strategy parameterizes this aspect of Knowledge Unit selection by indicating whether such remediation should occur as soon as the misconception is suspected, or wait until the current Knowledge Unit has been completed.

Control is achieved through information associated with each object which allows the system to respond dynamically to new tutoring situations. For instance, Knowledge Units, or topics represented as objects, have procedural "methods" associated with them that:

- teach their own topic interactively;
- explain knowledge didactically;
- teach their own prerequisites;
- test students for knowledge of that topic;
- summarize themselves;
- provide examples of their knowledge (an instantiation of a procedure or concept);
- provide motivation for a student learning the topic; and
- compare this knowledge with that of other Knowledge Units.

A specific tutoring strategy manifests itself by parameterizing the algorithm used to traverse the knowledge primitives network based on classifications of and relations between knowledge units. Several major strategies have thus far been implemented. For example, the tutor might always teach prerequisites before teaching the goal topic. Alternatively, it might provide a diagnostic probe to see if the student knows a topic. Prerequisites might be presented if the student doesn't exhibit enough knowledge on the probe. These prerequisites may be reached in various ways, such as depth-first and breadth-first traversal. An intermediate strategy is to specialize the prerequisite relation into "hard" prerequisites, which are always covered before the goal topic, and "soft" prerequisites, taught only when the student displays a deficiency.

Control and Reasoning about Examples. Another example of reasoning about tutoring primitives is shown by the actions of ExGen [17]; [13]. ExGen takes requests from the various components of the tutor and produces an example, question, or description of the concept being taught. A "seed" example base contains prototypical presentations of each type. ExGen's modification routine expands this into a much larger virtual space of presentations as needed. The goal is to enable the tutor to have flexibility in its presentation of examples and questions tasks that accompany those examples, without needing to represent all possible

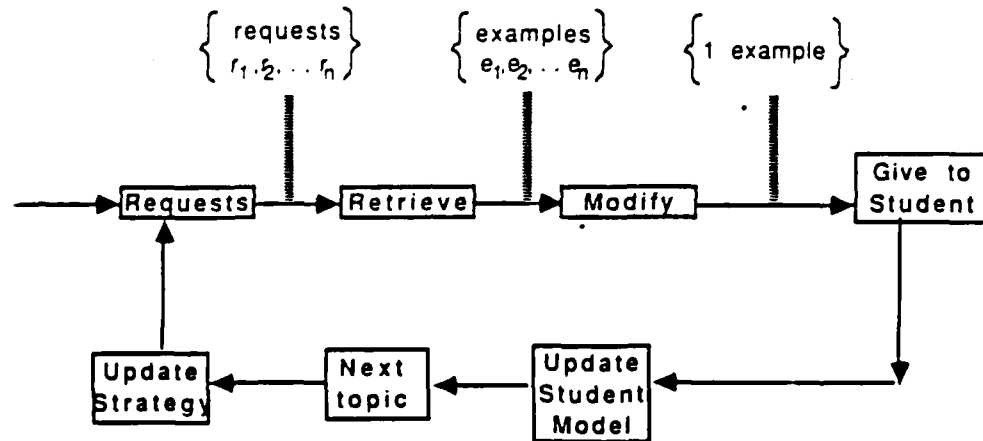


Figure 6: Reasoning About Examples.

presentations explicitly.

Requests given to ExGen are expressed as weighted constraints called requests (see Figure 6). The constraints are written in a language which describes logical combinations of the desired attributes of the example, and the weights on them represent the relative importance of each of these attributes. The returned example generally meets as many of the constraints as possible in the priority indicated by the weights.

ExGen is driven by *example generation specialists*, or knowledge sources, each of which examines the current discourse and student models and produces requests (weighted constraints) to be given to ExGen. These example generation specialists may be thought of as tutoring rules, encoding such general prescriptives as "when starting a new topic, give a start-up example", or "ask questions requiring a qualitative response before those involving quantities."

The tutoring strategy impacts on this layer of presentation selection by prioritizing the relative importance of the recommendations produced by each of the example generation specialists. Within a strategy, each specialist has a weight multiplied by the weight of the requests produced by the specialists. Altering the behavior of the presentation control is simply a matter of changing the weights on the specialists by selecting a new strategy.

For instance, one specialist requests that presentations describing the current Knowledge Unit be given, and another requests that the student be questioned. These competing requests

are ordered by the current tutoring strategy. We are also examining strategies for temporal ordering of the presentation of examples, such as Bridging Analogies ([2]; [8] unpublished) and Incremental Generalization.

Acquiring Tutoring Primitives Knowledge. Knowledge acquisition for tutoring primitives knowledge means encoding the questions, examples, analogies, and explanations that an expert might use to tutor a particular domain, as well as the reasoning he/she uses to decide how and when to use those primitives. We achieve knowledge acquisition for tutoring primitives through a graphical editor which is used by the instructional designer to encode and modify primitives and reasons why one primitive will be used instead of another. The graphical editor allows a teacher to generate and modify primitives without working in a programming language. The system presents a sheaf of "cards" listing a series of primitives. The user chooses a card and brings the primitive into an edit window, from which he/she can build new primitives.

3. TOOLS FOR REPRESENTING DISCOURSE KNOWLEDGE

Our tutors are beginning to represent and reason about their alternative responses to the student. Choices are concerned with how much information to give and what motivational comments to make. For instance, the machine must decide whether or not to:

- talk about the student's response;
- provide motivational feedback about the student's learning process;
- say whether an approach is appropriate, what a correct response is, and why a student's response is correct or incorrect; or
- provide hints, leading questions, or a counter-suggestion;

Motivational feedback may include asking questions about the student's interest in continuing or providing encouragement, congratulations, challenges, and other statements with affective or prelocutionary content. Control is modulated by which tutoring strategy is in effect, which in turn places constraints on what feedback or follow-up response to generate. The strategy may also specify that system action be predicated on whether the student's response was correct, or even that no response is to be given.

Reasoning about Discourse Level. As a start to this process we have defined several high-level response strategies and tactics (see Figure 7). For example, we have designated an informative response tactic as one in which the machine will elaborate, give reasons and congratulate the student. For each concept represented in the machine, some of these primitive responses are available and the machine will generate the requested tactic. However, we also advise the system about strategies such as Socratic tutoring, being brief, and being verbose.

RESPONSE STRATEGY		Verbose ②						10
		Brief ②						
		Socratic ②						
		Helpful ①						
TUTOR'S ACTION	echo answer		X		X	✓		
	encourage						✓	
	reveal answer			✓		X		
	congratulate	✓	X		X	X	✓	
	give reason	✓			X	X		
	challenge		X	X		✓	X	
	hints				X	✓		
	elaborate	✓			X			
		Informative	non-intrusive	directive	concise	coy	encouraging	
		RESPONSE TACTIC						

☒ Do
☒ Don't do
☐ Don't care

✓	Do
X	Don't do
	Don't care

Figure 7: Reasoning about Discourse Level.

Here we indicate a priority ordering; thus to be Socratic, the machine must place highest priority on the tactic called coy and secondary rating on the tactic to be informative. If there is a conflict between the checks and the crosses, that notation with the highest priority will win.

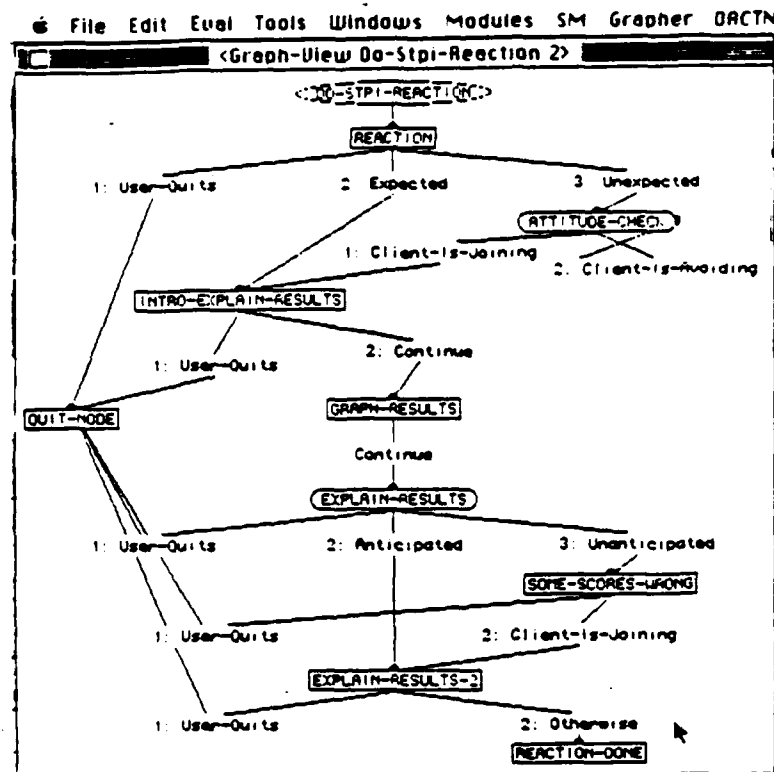
3.1 Managing Discourse

We realize that a more flexible and responsive discourse management technique is critical to any tutoring or consultant system. By discourse management, we mean the system's ability to maintain interactive discourse with the user and custom-tailor its responses beyond the generalized discourse level suggested above. Ideally, the system should tailor its response to the idiosyncracies of a particular user. Machine discourse and response need not be in natural language to be effective [10].

For example, the system should ensure that an intervention relates directly to an individual's personal history, learning style, and on-line experience with the system. It should dynamically reason about a user's actions, the curriculum, and the discourse history. In doing this the tutor should make each user feel that his/her unique situation has been responded to, appropriately and sensitively. In this way the system simulates one-on-one human tutoring behavior. The mechanism we use to do this is called a DACTN, *Discourse ACTION Transition Network*,⁵ which represents and controls the human-machine dialog. Figure 8 is a DACTN for responding to a user about the inventory test of questions that he/she took in the system described in Section 3.2. This graphic is taken directly off the screen of the system.

Sometimes the intervention steps designated by a DACTN are based on a taxonomy of fre-

⁵Rhymes with ACT-IN



11

Figure 8: Discourse Action Transition Network: DACTN.

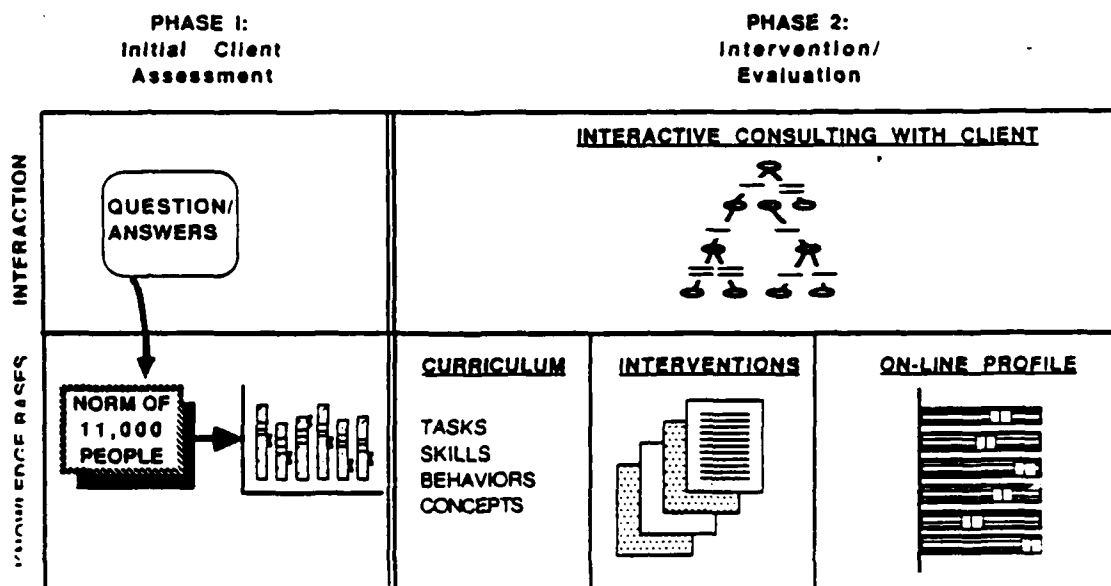


Figure 9: Two Phases of the Consultant

quently observed discourse sequences which provide default responses for the tutor [18]. The discourse manager also can reason about local context when making discourse decisions. Here local context is an aggregate of the client profile and response history.

The DACTN represents the space of possible discourse situations: Arcs track the state of the conversation and are defined as predicate sets while nodes provide actions for the tutor. The discourse manager first accesses the situation indicated by the arcs, resolving any conflicts between multiply-satisfied predicate sets, and then initiates the action indicated by the node at the termination of the satisfied arc.

Arcs represent discourse situations defined by sets of predicates over the client profile and the state of the system. For instance, the value of the arc "CLIENT-IS-AVOIDING" (top-half of Figure 8) is determined by inferring over the current state of the profile and recent client responses. Placing actions at the nodes rather than on the arcs, as was done in the ATN [15], allows nodes to represent abstract actions which can be expanded into concrete substeps when and if the node is reached during execution of the DACTN. For example, the node "EXPLAIN RESULTS" (middle of Figure 8) expands into yet another complete DACTN to be executed if this node is evaluated in the course of the intervention.

Each user response causes the user model, or in this case the personality profile, to be updated, which in turn affects the interpretation of the current discourse situation. DACTNs allow discourse control decisions to be based on a dynamic interpretation of the situation. In this way the mechanism remains flexible, domain-independent, and able to be dynamically rebuilt—decision points and machine actions are modifiable through a graphical editor, as explained in the Section 3.4. DACTNs have been implemented in two domains, one of which is described in the next section.

3.2 Example Discourse Knowledge

TEV (Time, Energy, and Vision) is a consultant tutor that presents interventions directed at improving an individual's personal time perspective. The system moves through two phases which model the human-to-human consultation process: 1) Initial Client Assessment, and 2) Intervention/Evaluation (see Figure 9).

Phase I: Initial Client Assessment.

During the first phase, TEV gathers information about the person's attitudes, knowledge, and skills using an assessment instrument which, in a non-computerized version, has been tested with 11,000 individuals. The client is presented with a series of statements which focus on attitudes, knowledge, and skills related to time perspective (Stanford Time Perspectives Inventory [5]). For each statement, the client is given a choice of five ratings ranging from very characteristic to very uncharacteristic and is asked to indicate "How characteristic is this of you?" Example statements include the following:

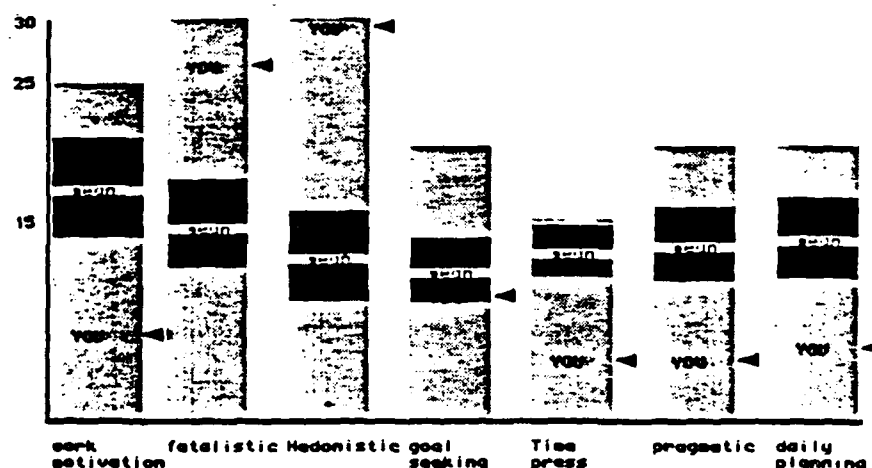


Figure 10: Actual Evaluation of a Client who Presented a Present-Oriented Time Perspective.

- Thinking about the future is pleasant to me.
- If things don't get done on time, I don't worry about them.
- I do things impulsively, making decisions on the spur of the moment.
- I try to live one day at a time.

Responses are evaluated according to the established interpretation of the instrument and then used in two ways. First they are shown to the client in a graph form based on the client's rating among his age-group peers (see Figure 10). Then an explanation of the results is provided along with an opportunity for the client to react (see Figure 11). The evaluation and the client's response are used as the initial database to form an internal client profile. The profile is to help determine the future direction of the consultation and to generate expectations about client responses. It becomes the primary determining knowledge base for designing the consultation and is constantly updated to provide a moment-to-moment view of the client.

Phase II: Intervention/Evaluation.

The second phase uses the profile to begin the consultation process. Here the goal is to create an ongoing dialog which leads the client to form and hold a more balanced time perspective. A second goal is to improve the client's repertoire of useful time management behaviors.

³The dialogue is not generated through natural language processing, rather through menu input and canned text output. This might be a limitation of the system. However, prior work in the area of computer-based therapy [Servan-Schreiber, 1986] shows that clinically significant therapy can be achieved in spite of this limitation.

Tutor: Your scores indicate that you have a basically present oriented time perspective. Do you agree?

Person A: No, I think I also consider the future in my life.

14

Tutor: Well, let's look more closely at your scores, and see where you think they are wrong.

Person A: OK.

Tutor: Your score on the hedonism scale indicates that you may be extremely similar to people who are primarily interested in having a good time, taking risks, and who often act impulsively. Does that sound like you?

Person A: Mostly, but it sounds a little extreme.

Tutor: OK, let's look now at the fatalism scale; it indicates that you may be very similar to people who live one day at a time in order to avoid planning for a future they believe is determined by fate. Is that what you do?

Person A: Yes, I guess so.

Tutor: Let's look at one more scale, Goal-seeking. Your score here shows that you may not be similar to people who plan for the future and get satisfaction from setting and achieving their goals. Do you agree?

Person A: It is true that I don't do much goal setting.

Tutor: It was mainly your scores on these scales that led me to view you as basically a present-oriented person. Since you disagree with the hedonism score slightly, why don't we say slightly present-oriented?

Person A: Yes, that sounds more like me.³

Figure 11: Dialogue about the Evaluation of Figure 10.

These goals are achieved through dialogue consultation, specifically through the use of interventions, which are exercises or presentations designed to facilitate awareness of time perspectives or to provide learning or practice of skills. Example interventions are "Learning to Say No," "Life Goals," and "Time Wasters." Dialogue strategies are derived from a large repertoire of similar activities used in one-on-one and group counseling over the last 15 years by experts in clinical psychology. These strategies and interventions have proven effective in improving time management skills for a large number of people.

TEV's orientation as a consultant tutor has led to a view of interventions as dialogs. Each intervention is seen as a distinct segment of an ongoing dialogue between TEV and the client which is extended by presentation of the next intervention. The consultation experience for each client is uniquely defined by the composite of high-level interventions and low-level discourse actions resulting from his/her responses to the system.

3.3 Representing Discourse Knowledge

In the system we have built, the consultant represents knowledge of discourse as alternative plans. Knowledge of alternative curriculum activities is stored as predefined plans and alternative discourse moves are stored as different plan contingencies in these prestored plans (see Figure 12). The consultant has limited planning ability to manage these plans and plan contingencies. Pedagogical activities and discourse knowledge have been articulated by a clinical psychologist and are used to generate the lesson plan in response to client input during the lesson

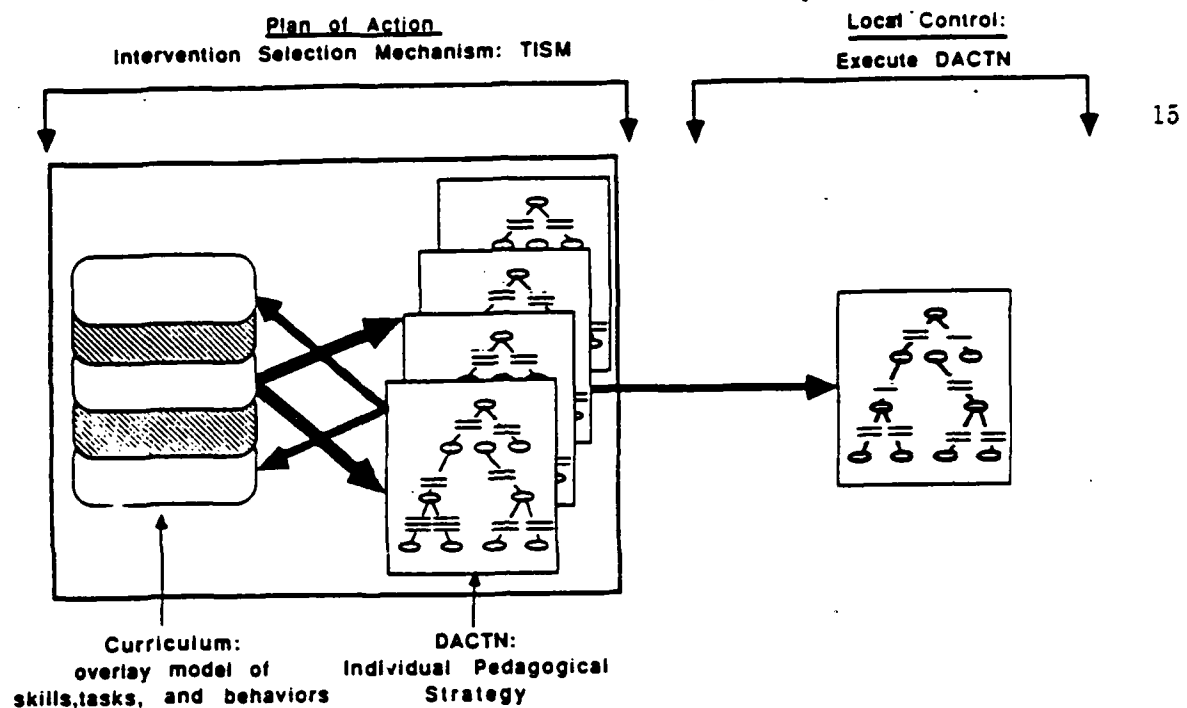


Figure 12: Levels of Control in TEV.

One characteristic aspect of the computational model of didactics is described here: the *plan of action* or lesson plan that enacts didactic operations. The *local context* in which a particular plan of action is triggered [14] was described in Section 3.1.

The plan of action is a unit of decision in the didactic process that manages knowledge about the curriculum, the available teaching resources, and the client's needs. In the case of a consultant, the curriculum consists of a prioritized overlay of skills, behaviors, and concepts which the client should be able to understand, demonstrate, and integrate into his/her lives. (For example, one of the skills the machine presents is to keep a 'to-do' list or to have the client state his/her priorities for the next month.) The plan of action is controlled by the TISM (Tev's Intervention Selection Mechanism) which models an expert's ability to select appropriate interventions for a specific student. For each instructional objective, several pedagogical approaches (DACTNS) are indicated as being able to achieve the chosen objective (see Figure 12). Alternatively, for each pedagogical approach, or single DACTN, several curriculum objectives might be achieved. Our experts have developed a library of resources to teach alternative curriculum items, such as identifying time-wasters. During a one-on-one consulting session, TISM chooses among these resources based on an understanding of the needs and learning style of the client.

These resources are represented in the consultant in the form of interventions. The system reasons about the current context in generating the next step in its plan of action. It is constrained by the client assessment, a record of the client's state of knowledge, and system history. The TISM is responsible for establishing a globally coherent instructional objective and for ensuring that curricula items follow each other in a way that matches the client's needs.

3.4 Acquiring Discourse Knowledge

Knowledge acquisition for discourse knowledge involves encoding the reasons why an instructor makes decisions about responding to the student and how he/she decides when such interventions will take place. We facilitate the knowledge acquisition process for discourse knowledge by use of a graphical editor in which the instructor selects interventions and modifies the dialogue "on-line." The editor facilitates piecewise development and evaluation of the system, thus providing an opportunity for a wide circle of people, including psychologists, teachers, curriculum developers, and instructional scientists, to participate in the process of system implementation.

Because DACTNs provide a structured framework for representing dialogs, we have been able to develop a visual dialogue editor which allows an expert to create new interventions graphically and have them automatically translated into LISP code. This allows the experts to work on knowledge acquisition without having to work with knowledge engineers. Thus we continue to elicit new interventions from our experts even as development and evaluation of TEV proceeds. By adding interventions to the library and linking them to the curriculum we expand TEV's repertoire without reworking the entire system.

The dialogue editor allows an expert to directly manipulate a graph of the dialogue where each question, statement, or action is represented in an editable node, and each arc (also editable) represents a discourse situation that could result from the client's response.

The expert adds a new question or statement and is led through a series of prompts designed to elicit the possible client responses. Each response has associated with it two pieces of information: a classification of the response, which is based on the current user profile, and the profile updates related to the choice of this response. Using a small set of classifications, i.e., EXPECTED, INDICATES-CONFUSION, AVOIDANCE, etc., the expert indicates his understanding of the meaning of this response. These classifications may depend on the current user profile, since this provides an indication of context. The profile modifications may include both updates based on the classification of the response and updates specific to this question and response.

As each question is added the graph is updated so the expert always has a view of the current state of the intervention. The underlying DACTN is created dynamically so that at any point in the editing it can be executed against default profiles, allowing the expert to check the appropriateness of the machine's responses.

4 TECHNOLOGY AND INTELLIGENT TUTORING SYSTEMS

4.1 Near-Term Goals

We are working toward the achievement of several near-term goals in this field. These are listed below:

- Real interaction and learning between the communities of instructional designers and builders of ITS.
- Wide expansion of ITS into new training and teaching areas as computer price goes down and memory and processing time goes up.
- Representing and reasoning about large numbers of teaching strategies based on advances in Artificial Intelligence, Discourse Methods, and Cognitive Science results about teaching and learning.
- Establishment of criteria by which domains can be assessed as potential applications for building new intelligent tutors. Such criteria include:
 - topics which are repeatedly taught to large groups, e.g., military domains;
 - difficult or dangerous domains;
 - domains in which there exists an ability (or willingness) of the organization to incorporate new system, new style.
- Distribution of ITS tools, including knowledge bases, control structures, among sites, between applications.

4.2 Long-Term Opportunities

We suggest that production of these systems will have long-term effects on education. In particular, we propose that the following will be possible:

- Establishment of distributed education. Training and literacy is now communicated by industry, community, schools, parents—education has moved out of the classroom. ITS will contribute to this process, enabling people to learn at any place or time, free of constraints.
- Enable students to become autonomous scholars. ITS will be used to access foreign knowledge bases, on-line encyclopedia, graphics, real-time, long-distance results (e.g., stock market closings), or communicate with other scholars and other knowledge bases.
- Establish human-machine partnership. Two intelligent entities will participate in joint problem solving. Each partner will use those features at which he/she it excels. Humans are best at intuitive thinking, reasoning from incomplete, uncertain knowledge, and using analogies from disparate fields to draw conclusions. Machines are best at memory retention, computation, pattern matching, making plans, and organizing tools.

- World-wide multi-media communication. Electronic networks, collision of media (television, cd rom, video, computer) and movies all use the same digital signal, leading to rapid and wide bandwidth communications.
- Construction of mega-scale knowledge bases, all recorded encyclopedic knowledge available at a keyboard.

5 REFERENCES

- [1] Atkins, T., *The Second Law*, Freedman, San Francisco, CA, 1982.
- [2] Brown, D., Clement, J., & Murray, T., Tutoring Specifications for a Computer Program Which Uses Analogies to Teach Mechanics, *Cognitive Processes Research Group Working Paper*. Department of Physics, University of Massachusetts, Amherst, MA, April 1986.
- [3] Duckworth, E., Kelley, J., & Wilson, S., "AI Goes to School," *it Academic Computing*, November, 1987.
- [4] Gonzalez, A., & Zimbardo, P., Time in Perspective, *Psychology Today*, pp. 21-26, March, 1985.
- [5] Gonzalez, A., & Zimbardo, P., *Stanford Time Perspective Inventory*, 1986.
- [6] McCalla, G., Greer, J., & the Scent Team, Intelligent Advising in Problem Solving Domains: The Scent-3 Architecture, *Proceedings of the International Conference on Intelligent Tutoring Systems*, University of Montreal, 1988.
- [7] MacMillian, S., Emme D., & Bekowitz, M., Instructional Planners: Lessons Learned, in Psotka, J., Massey, L.D., & Mutter, S., (Eds.), *Intelligent Tutoring Systems: Lessons Learned*, Lawrence Erlbaum Associates, Publishers, Hillsdale, NJ, 1988.
- [7] Murray, W., Control for Intelligent Tutoring Systems: A Comparison of Blackboard Architectures and Discourse Management Networks, *FMC Technical Reports #R-6267*, 1988.
- [8] Murray, T., Schultz, K., Clement, J., & Brown, D., "Dealing with Science Misconceptions Using an Analogy Based Computer Program," unpublished document.
- [9] Reichman, R., *Making Computers Talk Like You and Me*, MIT Press, 1985.
- [10] Servan-Schreiber, D., Artificial Intelligence in Psychiatry, *Journal of Nervous and Mental Disease*, 174, pp. 191-202, 1983.
- [11] Servan-Schreiber, D., From Intelligent Tutoring to Computerized Psychotherapy, *Proceedings of the Sixth National Conference on Artificial Intelligence*, pp. 66-71, 1987.
- [12] Slovin, T., & Woolf, B. P., A Consultant Tutor for Personal Development. *Proceedings of the International Conference on Intelligent Tutoring Systems*. University of Montreal. 1988.
- [13] Suthers, D., & Rissland, E., Ex Gen: A Constraint Satisfying Example Generator. *Cornell Technical Report # 88-71*. University of Massachusetts, Amherst, MA. 1988.

- [14] Wenger, E., *Artificial Intelligence and Tutoring Systems*, Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1988.
- [15] Woods, W., Transition Network Grammars for Natural Language Analysis, *Communications of the ACM*, Vol 13:10, pp. 591-606, 1970.
- [16] Woolf, B., & Cunningham, P., Multiple Knowledge Sources in Intelligent Tutoring Systems, in *IEEE Expert*, Summer, 1987.
- [17] Woolf, B., Suthers, D., & Murray, T., Discourse Control for Tutoring: Case Studies in Example Generation, to be published as a COINS Tech Report, University of Massachusetts, Amherst, MA.
- [18] Woolf, B., & Murray, T., A Framework for Representing Tutorial Discourse, *International Joint Conference in Artificial Intelligence (IJCAI-87)*, Morgan Kaufmann, Inc., Los Altos, CA, 1987.

DESIGN OF A DOMAIN-INDEPENDENT PROBLEM SOLVING
INSTRUCTIONAL STRATEGY FOR
INTELLIGENT COMPUTER-ASSISTED INSTRUCTION^{1,2,3}

Harold F. O'Neil, Jr.
University of Southern California and Advance Design Information, Inc.

Dean A. Slawson
University of California, Los Angeles, and Advance Design Information, Inc.

Eva L. Baker
UCLA Center for the Study of Evaluation and Advance Design Information, Inc.

SUMMARY

The document is organized in three sections. Section I is included to provide background and context. Section II contains the domain-independent instructional strategy (Tables 2 and 3) and documents the evolutionary changes in our thinking on this strategy. The information in this section was derived from several sources: the research literature, knowledge elicitation of an Army subject matter expert (SME) and expert teacher, as well as from reviews of draft strategies by ADI consultants, particularly Drs. Robert Gagne, Douglas Towne, and Richard Clark. Significant contributions were also made by Dr. Zhongmin Li. In addition, relevant comments by colleagues at Perceptronics and Harris Corporation have been incorporated. Finally, relevant suggestions by government monitors on our contract research project were implemented. In Section III, implementation issues are discussed and future developments suggested.⁴

BACKGROUND AND CONTEXT

The ideas for this chapter were derived from a program of research in the area of intelligent computer-assisted instruction (ICAI) with which we were associated. We view the application of ICAI as one way of increasing the cost-effectiveness of future education and training systems. We prefer the term ICAI to intelligent tutoring systems (ITS) because we view tutoring as only one of many possible

¹The research reported herein was supported in part by the Air Force Human Resources Laboratory, Army Research Institute for the Behavioral and Social Sciences, Navy Training Systems Center, and Advance Design Information, Inc. However, the views, opinions, and/or findings contained in this report are the authors', and should not be construed as an official Department position, policy, or decision, unless so designated by other official documentation.

²A version of this paper will be presented at the 1989 Knowledge Architectures in Intelligent Tutoring Systems Conference, San Antonio, Texas.

³A version of this paper will be published as a chapter in Burns, H., Luckhardt, C., & Ratlett, J., (Eds.) *Knowledge architectures in intelligent tutoring systems*, Orlando, Florida, Academic Press, 1990.

⁴The chapter version of this paper will focus on examples of instructional rules for a particular environment, i.e., teaching selected troubleshooting task strategies of the electrical system of the Improved TOW Vehicle (ITV). It will also document representative domain specific instantiations of principles, procedures, concepts, and facts.

instructional strategies. ICAI also offers a technology to implement one-on-one tutoring, which Bloom (1984) suggests is the most effective educational intervention. In particular, we were interested in designing and developing software tools that would make the design and development process of ICAI more efficient and effective. From our viewpoint in the instructional arena, we originally thought that this would provide both domain-independent and domain-dependent instructional strategies. Thus, each strategy would be instantiated in the domain of interest. These strategies would collectively teach the subject matter in question. This chapter will briefly discuss intelligent computer-assisted instruction and then focus on what we consider the key technical issues.

Intelligent Computer-Assisted Instruction

ICAI is the application of artificial intelligence to computer-assisted instruction. Artificial intelligence, a branch of computer science, is defined as making computers smart in order to (a) make them more useful and (b) better understand human intelligence (Winston, 1977). Topic areas in artificial intelligence have included natural language processing, vision, knowledge representation, spoken language, planning, and the development of expert systems.

Expert system technology is the branch of artificial intelligence which is, at this point, most relevant to ICAI. ICAI systems use approaches from artificial intelligence and cognitive science to teach a range of subject matters. Representative types of subjects include: computer programming in PROUST (Johnson & Soloway, 1983, 1987) or the LISP Tutor (Anderson, Boyle, & Reiser, 1985), rules in ALGEBRA (McArthur, Stasz & Hotta, 1987), and diagnosis of infectious diseases in GUIDON (Clancey, 1979, 1987). Representative research in ICAI is described by O'Neil, Anderson and Freeman (1986) and Wenger (1987).

Progress in cognitive science has been made in the following areas: identification and analysis of misconceptions or "bugs" (Clement, Lockhead, & Soloway, 1980), the use of learning strategies (O'Neil & Spielberger, 1979; Weinstein & Mayer, 1986), the nature of expertise as expert versus novice knowledge (Chi, Glaser, & Rees, 1982), the role of mental models in learning (Kieras & Bovair, 1983, Konoske & Ellis, 1986), and the role of self-explanations in problem solving (Chi, Bassok, Lewis, Reimann, & Glaser, 1987).

The key components of an ICAI system consist of: (a) the knowledge base—that is, what the student is to learn, i.e., the expert model which represents both the relevant knowledge in the domain and can solve problems as an expert based on this knowledge; (b) a student model, in which a model is constructed by comparing the student's performance to the computer-based expert's behavior on the same task; and finally (c) a tutor, that is, instructional techniques for teaching the declarative or procedural knowledge. This final component represents the teacher and must be able to apply the appropriate instructional tactics at the appropriate times. It should model the desirable properties of a human tutor. In general, the tutor must know what to say to the learner and when to say it. In addition, it must know how to take the learner from one stage of skill to another and how to help the learner, given his or her current state of knowledge. In general, there are few extant examples of complete ICAI systems.

Although suggestive evidence has been provided by Anderson, Boyle, and Reiser (1985) and Baker, Aschbacher, and Feifer (1985), few of these ICAI projects have been evaluated in any rigorous fashion. There are no examples of the explicit use of formative evaluation. In a sense they have all been toy systems for research and demonstration. Nonetheless these projects have generated a good deal of excitement and enthusiasm because they indicate that ICAI systems can be effective instructional environments.

However, few instructional design considerations (e.g., Ellis, Wulfek & Fredericks, 1979; Park, Perez & Seidel, 1987; or Reigeluth, 1987) are reflected in ICAI tutors. An exception is the work of Baker et al. (1985), which suggests instructional strategies to improve existing ICAI programs. This chapter reports a systematic attempt to provide an instructional framework for the design of an ICAI system. In particular, we have struggled with the specifications of domain-independent instructional strategies and domain-dependent instructional strategies.

CONCEPTUAL ISSUES

A particularly knotty problem associated with developing the framework for domain-independent instructional strategies has been determining the boundaries of concepts such as "domain" and "independent."

What's a "Domain"? Part of the issue revolves around the definition of *domain*. In cognitive science, the term usually refers to a subject matter area (e.g., math) or a process within a subject matter area (e.g., use math formulae). The term *domain* is often used synonymously with application area (e.g., equipment maintenance). Further, within an application area the focus has often been on the task level (e.g., diagnose and troubleshoot an electrical system). Instructional researchers, however, tend to use the term *domain* to refer to performance outcomes, e.g., the learning of procedures. Thus, *domain* could legitimately refer to subject matter, application area, task, or performance outcome. We believe it is essential to be specific, explicit, and consistent in our meanings of critical terms. We will use the term *domain* to refer either to task or to subject matter area.

What's "Independent"? Another part of the issue revolves around the term *independent* (i.e., independent of what?). Given the diversity of definitions of domain, *independent* can have a number of interpretations. *Independent* can mean "other than" or "as well as" the target domain. Domain-independent instructional strategies could mean independent of the subject matter (i.e., strategies independent of math), or independent of the performance outcome (i.e., strategies appropriate both to concept learning and to problem solving), or both. The problem of domain independence, in other words, is a problem of degree of transfer or generalizability.

Our solution to this issue is based on two key assumptions: (a) it is mandatory for instructional practices to specify for the designer the outcome or objective of the learning (e.g., problem solving); and (b) various outcomes (e.g., problem solving or principle learning) have specific learning conditions, i.e., instructional strategies. (The second assumption was adopted from the framework of Bob Gagné and David Merrill.) These learning conditions are dependent on outcome (e.g., problem solving), but are independent of domain or task as we are using it (e.g., diagnose and troubleshoot the electrical system). These learning conditions or strategies are also thought to be independent of subject matter or application area. Nonetheless, these conditions (or strategies) must be instantiated for a particular domain.

What's "Dependent"? Reflecting the problem of independence is the issue of domain dependence. Our analysis suggests that *domain dependence* indicates a domain that is exclusive or unique to the particular task area, with no generalizability across either subject matter or types of performance outcomes.

The next set of discriminations involve distinguishing between instantiations of domain-independent instructional strategies and domain-dependent instructional strategies. After considerable effort to develop counter examples, we have agreed that we have found no domain-dependent instructional strategies that could not just as well be called domain-specific instantiations of domain-independent instructional strategies. This finding may counter some work on problem solving in cognitive science. However, we believe that while *task* requirements differ enormously, task-specific instruction is best conceived of as instantiations of domain-independent instructional strategies rather than as domain-dependent instructional strategies unique to the particular domain. In fact, the differences among domain-dependent and domain-specific instantiations of strategies are not hard and fast, and, undoubtedly, share space on the same continuum.

With respect to our rule classification as domain-dependent or domain-independent, we have decided to operationally define *independent* in terms of a strategy's appropriateness to a particular class of learning outcomes, specifically, problem solving.

Domain Independent Instructional Strategy

For our own work we have chosen problem solving as it is appropriate for ICAI technology. "Problem Solving" (a Gagné term) is roughly equivalent to Merrill's "Using Principles." The domain-independent instructional strategies (or, in Gagné's terms, learning conditions) for problem solving are shown in Table 1.

-
1. Retrieval of relevant rules and concepts
 2. Successive presentation of novel problem situations
 3. Demonstration of solutions by student

(Gagne, 1977)

Table 1. Domain-Independent Instructional Strategies
(Learning Conditions) to Teach Problem Solving Outcomes

The instructional strategies in Table 1 are clearly at a very general level. For this research program our area of interest is troubleshooting. Troubleshooting is a concept that comprises many activities; one activity is problem solving. The technician is confronted with a malfunction which shows itself in certain "symptoms." He or she must then use his or her knowledge of the system, the tools available for diagnosis, and troubleshooting strategies to find the source(s) of the malfunction. Each malfunction with its set of symptoms constitutes a new problem to be solved by a thinking process. The more rational and informed the thinking can be, the more efficient the procedure of locating the trouble will be. We have provided the instructional strategies to teach diagnostic problem solving in Table 2.⁵

In Table 2, the domain-independent instructional sequence is provided in list format since this is clearer and more efficient than rule format for communicating macro sequences. The rules which sequence items in this list are "housekeeping"—not instructional—and would be devised by the implementor. The domain-independent strategy in Table 2 was instantiated in the area of troubleshooting for the turret system of the Improved Tow Vehicle (ITV). Before we provide the next level of detail of these strategies in troubleshooting in Table 3, we will discuss the troubleshooting area itself.

Troubleshooting is a broad area of activity. We have chosen to focus on the subset of tasks that include the use of troubleshooting aids and techniques to test hypotheses and locate and correct faults.

Carrying out troubleshooting in this area involves following procedures of several kinds. Sometimes the trouble can actually be found by following a fixed sequence of steps (this is one end of a continuum of complexity of troubleshooting tasks). More often, though, procedures are simply parts of the whole activity that is required. There are procedures for identifying the location of parts of the system. There are procedures for taking apart equipment components. And there are procedures for using test equipment. All of these are procedures that must be well known by the technician, else he or she will be impeded in his or her main task—thinking out the solution of a problem.

Presumably, an expert tutor would break down troubleshooting into at least five components:

1. Identifying equipment parts by name (for matching with technical manuals, supply lists);
2. Identifying the flow of power and/or current through the system;
3. Demonstrating the function of each part in terms of the flow;
4. Demonstrating the use of test equipment; and
5. Strategies for troubleshooting.

It should be emphasized particularly that this knowledge is predominantly what is called "procedural." However, declarative knowledge (such as the equipment "lore" of the veteran) is useful as an aid to encoding and retrieval of the procedures themselves.

⁵ This strategy is based on a refinement of Dr. Robert M. Gagne's work in the teaching of troubleshooting.

1. Define diagnostic problem solving family to be taught by describing problem solving characteristics or events pertaining to the family (e.g. "Electrical troubleshooting consists of checking current flow at input and output points, etc...").
2. Communicate a description of the appropriate example device(s), as whole systems, and necessary concepts and principles of operation in the order prescribed below:
 - a. Present name and brief overall description of the device including operational controls or inputs and ways in which failures are indicated on the device w/ examples.
 - b. Present concepts of causal media in system (e.g. current flow if electrical system, forces if mechanical, etc.)
 - c. Present concepts of schematic representation and methods of illustrating physical/schematic mapping to be used, with examples.
 - d. Present concept of replaceable units. Present names and locations and functions of replaceable units--using diagrams as needed--as examples. Require the student to identify the parts by pointing to them.
 - e. Teach operation and function (how the system works in terms of inputs, controls, component functions, causal flows, and outputs) of major activating and intermediary components, with reference to schematics and physical layout diagrams, taught by tracing causal paths through components on schematics and referencing physical locations.
3. Confirm or teach subordinate skills (e.g., demonstrating how each check is made, how each exchange is done in the example device, etc.)
4. Describe and demonstrate appropriate diagnostic problem-solving task strategies to be taught for this application (e.g. select component to test based on malfunction probability, etc.).
5. Provide practice, using a variety of novel problems requiring the strategies taught, and provide feedback and correction.

Table 2. Domain Independent Instructional Strategy to Teach Diagnostic Problem-Solving

Table 3 was generated by ADI consultant Dr. Robert Gagne. Entries 1, 2, and 3 are domain-specific instantiations of the learning condition "retrieval of relevant rules and concepts" (see Table 1). Entry 4 is the "successive presentations of novel problem solutions" (see Table 1). And Entry 5 is the "demonstration of solutions by student" (see Table 1).

As is shown in Table 3, in the application area of ITV, the domain-specific problem-solving task strategies are to find system faults based on: (a) malfunction probability, (b) change cost, and (c) split-half search. These are taught using domain-specific instantiations of domain-independent strategies.

In summary, the instructional outline in Table 3 presents a domain-independent *instructional* strategy (Entries 1-5) for teaching the domain-dependent problem solving *task* strategy (e.g., split-half) of electrical troubleshooting. We view the teaching of these task strategies as "using principles": Tasks would be taught using a modified version of Merrill's Component Display Theory (Merrill, 1987). These tasks are taught using domain-specific instantiations of domain-independent instructional strategies. Our project provided a first cut of the instantiations of these strategies by providing instantiations of the domain-independent instructional strategies in troubleshooting the electrical faults of a particular vehicle.

To teach domain-independent troubleshooting, using several malfunctions of the ITV turret system as instances of domain-independent strategies:

1. Communicate a description of the system as a whole to identify schema, having the following components.
 - a. concepts of current flow in each subsystem
 - b. concepts of major replaceable parts by name
 - c. concepts of schematic representations of parts
 - d. functions of parts
 2. Communicate that troubleshooting consists of checking current flow at input and output points of parts, and then replacing the part that shows an absence of proper output.
 3. Confirm or teach subordinate skills, demonstrating how each check is made and how each replacement is done.
 4. Describe and demonstrate the problem-solving task strategies:
 - a. malfunction probability
 - b. change cost
 - c. split-half technique
 5. Provide practice, using a variety of problems requiring troubleshooting with the use of these strategies, and providing feedback and correction.
-

Table 3. Domain-Independent Instructional Strategy to Teach Electronic Troubleshooting

Revisions and Instantiations

An Army ITV Electronics Troubleshooting SME met with Perceptronics and ADI personnel for three days for elicitation of instantiations of instructional and remedial strategies and elicitation of student errors and deficiencies. The SME, working with ADI, suggested changes in the order and content of instructional strategies; these are reflected in the Tables 2a and 3a, which are revisions of Tables 2 and 3, respectively. Although this document reflects interactions with the SME, it has not been reviewed or approved by him. The tables were also reviewed by colleagues at Perceptronics and Harris Corporation but no needed changes were noted.

(SME comments or changes in italics)

1. Define diagnostic problem solving family to be taught by describing problem solving characteristics or events pertaining to the family (*no changes*)
2. Communicate a description of the appropriate example device(s), as whole systems, and necessary concepts and principles of operation in the order prescribed below:
 - a. Present name and brief overall description of the device including operational controls or inputs and ways in which failures are indicated on the device w. examples.

- b. Present concepts of *how things are caused* in the system (e.g. *data/signal* flow if electrical system, forces if mechanical, etc.) (revised terms)
 - c. Present concepts of schematic representation and methods of illustrating physical/schematic mapping to be used; examples.
 - d. Present concept *and function* of replaceable units--using *block* diagrams as needed--as examples. Require the student to identify the parts by pointing to them. (*Teach top level description first*)
 - e. Teach operation and function (how the system works in terms of inputs, controls, component functions, causal flows, and outputs) of major activating and intermediary components, with reference to schematics and physical layout diagrams, taught by tracing causal paths through components on schematics and referencing physical locations. (*Teach detailed operation and function using schematics after general function of replaceable units in previous presentation*)
3. Confirm or teach subordinate skills (*no changes*)
 4. Describe and demonstrate appropriate diagnostic problem solving task strategies to be taught for this application (*no changes*)
 5. Provide practice, using a variety of novel problems requiring the strategies taught and provide feedback and corrections (*no changes*)

Table 2a. Revised Domain Independent Instructional Strategy to
Teach Diagnostic Problem-Solving

(SME comments or changes in italics)

1. *Communicate that "Electrical troubleshooting consists of using techniques and strategies to verify data or signal flow in a device in order to find and replace faulty LRU's (line replaceable units)". (Changed in wording, order -- was step 2)*
2. Communicate a description of the example system as a whole to identify schema, having the following components: (*Note change in order and grouping of substeps; also, as prerequisites, these concepts would apply for this application in the pretest only, before any instruction.*)
 - a. *Present an orientation which typically includes demonstration and (sometimes) hands on experience with the system in full operation including operational controls or inputs and ways in which failures are indicated on the device (with examples). With experienced students, present new things by building on previous knowledge, highlighting similarities and differences.*
 - b. *Use of reference designators and concept of line replaceable units (LRU's). Briefly present names, locations, and functions (generally) of specific LRU's to be used in this instruction--using block diagrams, TM, etc.--as examples. Require the student to identify the parts by function by pointing to them. In this context, present concepts of how things are caused in the system by major subsystem functions and data/signal flows. (In classroom setting, this is often a brief presentation followed by hands-on operation.)*

- c. *Present concepts of schematic representations of systems and methods of illustrating physical/schematic mapping to be used, with examples. (SME says some kind of diagrams would be helpful for physical/schematic mapping but in the school they learn where parts are on vehicle by hands-on experience, not during prerequisite presentation. The level of presentation would be at the level of modules or block diagrams before detailed schematics, e.g. "there is 24 VDC between these pins connecting these two LRU's")*
- d. *Teach detailed operation and function (how the system works in terms of inputs, controls, component functions, causal flows, and outputs) of major activating and intermediary components, with reference to schematics and physical layout diagrams, taught by tracing data or signal paths through components on schematics and referencing physical locations. (At this point the SME is working at the level of specific components inside a module. Also, any peculiarities of schematic representation for this vehicle would be taught in the current context.)*
3. *Confirm or teach subordinate skills (Again, since these are prerequisites, the test of these skills would precede the course and they will not actually be taught here in the demo application. Included are basic troubleshooting knowledge, safety, part replacement, standard operating procedures, etc.)*
4. *Describe and demonstrate the troubleshooting task strategies. These are taught in the order of typical use or most likely frequency of use, except that split half is taught first because the other strategies are frequently combined with this "default" strategy. Practice should be provided after each strategy, including examples which integrate and require discriminations between the strategies taught up to that point. The strategies to be taught assist in the basic troubleshooting process of localizing, isolating, and identifying faults:*
 - a. *split half (basic strategy for determining efficient fault search in series circuits)*
 - b. *malfunction probability (test likely failed components first)*
 - c. *change cost (e.g. perform easiest tests first)*
 - d. *necessary and sufficient factors (using logic and knowledge of normal parameters to localize and identify faults)*
 - e. *commonality of fault (using principle that concurrent symptoms with common component imply the common component is faulty)*
5. *Provide practice, using a variety of novel problems requiring the strategies taught and provide feedback and corrections. The final practice should integrate all techniques taught. After the techniques are practiced, a posttest requires the learner to locate faults giving reasons for steps to verify understanding.*

Table 3a. Revised Domain-Independent Instructional Strategy to
Teach Electronic Troubleshooting

IMPLEMENTATION ISSUES

Students would be taught electronics troubleshooting task strategies using the instructional strategy found in Table 3a (Entries 1, 4, and 5). Since the application is to teach troubleshooting task strategies, a pretest would be needed to screen out any students who do not have the prerequisite facts, concepts, and

skills (Table 3a: Entries 2 and 3). Students experienced in troubleshooting the ITV should get a perfect pretest score, barring error. The pretest is not part of the instructional rules or instantiations for teaching the troubleshooting task strategies, and so is not included in our designed knowledge base. Likewise, rules for remediation of basic skills and knowledge and for posttesting are not included. However, in the chapter version of this document, instantiations of the instructional strategies will be provided for the communication of selected troubleshooting task strategies (e.g., presentation of the principles with examples and practice). In addition, remediation to correct misunderstanding or misapplication of the task strategies themselves, plus a few deficiencies common to even experienced troubleshooters (e.g., inability to locate components), will be selectively provided.

The domains of interest will be portions of three ITV troubleshooting cases elicited by Perceptronics in a modified petri net formalism, properly contexted and, with the assertion of various faults, used as examples and practice exercises for teaching the use of some of the troubleshooting task strategies. The task strategies that are adequately covered in the malfunction networks are "fault search based on malfunction probability" and "change cost" (Table 3a: Entries 4b and 4c). The other task strategies would generally be taught in the same manner, but are not included as they did not appear in at least two malfunctions.

An implementor would note the implicit macro sequencing information in Table 3a. Concepts or principles are taught in the order in which they appear. Micro sequencing for teaching the task strategies is indicated in specific rules. At a micro level, the presentation of each troubleshooting task strategy is followed by practice on the strategy just taught and instruction and practice on synthesizing the new knowledge with any previously taught strategies.

Future Developments

Following are some specific areas in which additional work is needed for the researcher and for the implementor.

Alternate and Parameterized Strategies. One of the requirements for a researcher is to be able to modify and try out variations of the domain-independent instructional strategies, i.e., the provision of "knobs and dials." For example, the researcher might vary: (a) the degree or kind of learner control, (b) the immediacy, amount, or kind of feedback (e.g., guided practice option), (c) performance criteria or branching options after learner performance evaluation (e.g., if fail then ignore/retry/kick out), and (d) the amount or kind of tutorial help or explanations available when requested by learner.

Any of these options could be implemented in a variety of ways. For example, learner control could be optionally "off," "on," or "automatic" (i.e., under control of the tutor). With or without control by the researcher, the default or "automatic" selection of alternative strategies must be specified by metarules which determine the active rule sets.

Review Techniques. In addition to the areas of knowledge base refinement indicated in the comments above, general refinement of the knowledge communications, rules, and rule classifications are needed. Depending on the degree to which an application is to be developed (i.e., the courseware issue), further work may also be required in specifying rules that are specific to a learner interface. The rules will have to undergo more extensive validation in the areas of consistency and completeness.

Since we were required by contract to produce no software, our series of reviews focused on determining the instructional validity of the rules and the completeness of the rule set, and on providing domain-specific instantiations. Draft rules were reviewed by ADI consultants Drs. Douglas Towne and Richard Clark, and were also reviewed in a government IPR. Suggestions were also made by Perceptronics and a meeting was held with Harris colleagues to discuss the rule set.

Validation included checking for completeness, consistency, and correctness of the rules. Simulations (via people, not computer software) of rule firing under error-free instruction resulted in rule changes. Further, the knowledge communications were revised so that they now have labels for their constituent parts (a Perceptronics suggestion). These labels would allow the domain-independent sequencing rules to reference displays indirectly, and the instantiations with explicitly coded sequencing of presentations would be eliminated. ADI's check for consistency and completeness also turned up some inadequate rules, rule conflicts, and misclassifications, which were resolved and which resulted in a modified rule set.

Where Are We Now?

We have a good "first cut" of a domain-independent instructional strategy to teach troubleshooting of an electrical system. Further, we have a reasonable rule set to teach one aspect of domain-specific task strategies. Our next step is to implement our design and test it. We are currently seeking funding to accomplish this goal.

REFERENCES

- Anderson, R.J., Boyle, C.F., & Reiser, B.J. (1985). Intelligent tutoring systems. *Science*, 228, 456-462.
- Baker, E.L., Bradley, C., Aschbacher, P., & Feifer, R. (1985). *Intelligent computer-assisted instruction (ICAI) study*. Final Report to Jet Propulsion Laboratory. Los Angeles: UCLA Center for the Study of Evaluation.
- Chi, M.T.H., Bassok, M., Lewis, M.W., Reimann, P., & Glaser, R. (1987 November). *Self-explanations: How students study and use examples in learning to solve problems* (Technical Report No. 9). Pittsburgh, PA: University of Pittsburgh, Learning Research and Development Center.
- Chi, M.T.H., Glaser, R., & Rees, E. (1982). Expertise in problem solving. In R.J. Sternberg (Ed.), *Advances in the psychology of human intelligence* (Vol. 1, pp. 7-76). Hillsdale, NJ: Erlbaum.
- Clancey, W.J. (1979). *Transfer of rule-based expertise through tutorial dialogue* (Report No. CS-769). Stanford, CA: Computer Science Department, Stanford University.
- Clancey, W.J. (1987). *Knowledge-based tutoring/The GUIDON program*. Cambridge, MA: MIT Press.
- Clement, J., Lockhead, J., & Soloway, E. (1980). *Positive effects of computer programming on students' understanding of variables and equations*. In Proceedings of the National Association for Computing Machinery, Nashville.
- Ellis, J., Wulfeck, W.H., & Fredericks, P.S. (1979). *The instructional quality inventory: II. User's manual* (NPRDC SR 79-24). San Diego, CA: Navy Personnel Research and Development Center. (AD-A083-678).
- Gagné, R.M. (1977). *The conditions of learning* (3rd ed.). New York: Holt, Rinehart & Winston.
- Johnson, W.L., & Soloway, E. (1983). *PROUST: Knowledge-based program understanding* (Report No. 285). New Haven: Computer Science Department, Yale University.
- Johnson, W.L., & Soloway, E. (1987). PROUST: An automatic debugger for Pascal programs. In G.P. Kearsley (Ed.), *Artificial intelligence: Applications and methodology*. Redding, MA: Addison-Wesley.
- Kieras, D.E., & Bovair, S. (March 1983). *The role of a mental model in learning to operate a device* (Technical Report No. 13 RZ/DP/TR-83/ONR-13). Tucson: University of Arizona, Department of Psychology.
- Konoske, P.J., & Ellis, J.A. (1986, December). *Cognitive factors in learning and retention of procedural tasks* (NPRDC TR 87-14). San Diego, CA: Navy Personnel Research and Development Center.
- McArthur, D., Stasz, C., & Hotta, J. (1987). Learning problem-solving skills in algebra. *Journal of Educational Technology Systems*, 15(3), 303-324.

- Merrill, M.D. (1987). A lesson based on the component display theory. In C.M. Reigeluth (Ed.), *Instructional theories in action*. Hillsdale, NJ: Lawrence Erlbaum.
- O'Neil, H.F., Jr., & Spielberger, C.D. (Eds.). (1979). *Cognitive and affective learning strategies*. New York: Academic Press.
- O'Neil, H.F., Jr., Anderson, C.L., & Freeman, J.A. (1986). Research in teaching in the Armed Forces. In M.C. Wittrock (Ed.), *Handbook of research on teaching* (3rd ed.). New York: Macmillan.
- Park, P., Perez, R.S., & Seidel, R.J. (1987). Intelligent CAI: Old wine in new bottles or a new vintage? In G.P. Kearsley (Ed.), *Artificial intelligence: Applications and methodology*. Reading, MA: Addison-Wesley.
- Reigeluth, C.M. (Ed.) (1987). *Instructional theories in action: Lessons illustrating selected theories and models*. Hillsdale, NJ: Lawrence Erlbaum.
- Weinstein, C.F., & Mayer, R.F. (1986). The teaching of learning strategies. In M.C. Wittrock (Ed.), *Handbook of research on teaching* (3rd ed.). New York: Macmillan.
- Wenger, E. (1987). *Artificial intelligence and tutoring systems*. Los Altos, CA: Morgan Kaufmann.

MANAGING COMMUNICATION KNOWLEDGE

Kathleen M. Swigger
Computer Science Department
University of North Texas
Denton, Texas 76203

ABSTRACT

Intelligent tutoring systems which are capable of training students in complex problem solving tasks require man/machine interfaces that are extremely flexible. This paper discusses the issues surrounding the design, implementation, and evaluation of flexible training environments.

This paper will begin by defining student/computer interaction and will explain how this definition is becoming obsolete as we move toward more student-centered, reactive environments. The idea of microworlds, similar to the Orbital Mechanics tutor, demonstrates the effectiveness of these new student-centered environments. Because advances in interface design have allowed us to explore more complex reasoning tasks, researchers should now focus on the question of developing more formal design specification tools that can be used to describe a particular type of interaction at a conceptual, rather than implementation level. Formal specification techniques are particularly useful because they describe user behavior, independent of software implementation. Finally, this paper explores the question of how to evaluate student/computer interfaces, particularly reactive learning environments.

INTRODUCTION

This paper describes the communication knowledge part in an intelligent tutoring system (ITS). Traditionally, communication knowledge has been defined as a specific module that is responsible for administering the interaction between the student and computer [27]. Under this narrow definition, the communication module can be considered an information exchange system. More recently, the definition of communication knowledge has been expanded to include the entire tutoring system. Indeed, Wenger [25] maintains that the primary purpose of an intelligent tutoring system is to provide the student with a set of operators that will cause/or support the communication of knowledge. These set of operators now include a vast array of interactive styles (menus, natural language, icons) and interactive devices (mouse, touch pads, speech recognition). Under this broader definition, the designer of intelligent tutors is responsible for building massive communication management system that controls and monitors a student's learning environment.

The following paper will discuss how designers of intelligent tutoring systems confront the problem of creating systems that communicate rather than simply provide information; the distinction between communication management systems and information exchange systems is fundamental to this discussion. Further, this paper will attempt to answer such questions as; What is communication knowledge in an intelligent tutoring system? How is it different from other types of communication? How does one represent and present communication knowledge? and How does one evaluate an effective communication system? These and other issues will be discussed below.

The first section explains the relationship between communication as defined in intelligent tutoring systems and theories of communication as developed by human communication researchers. The second section defines communication knowledge within the context of existing intelligent tutors. The third section discusses a methodology that can, hopefully, assist designers of communication knowledge. The fourth section discusses the question of evaluating the communication between student and tutor. Finally, the paper concludes with a discussion of future research.

A DEFINITION OF COMMUNICATION

The field of human communication studies has a body of well-established knowledge about which intelligent tutor interface designers are largely unaware. The term communication has had a variety of meanings for different communication theorists. Its most restrictive definition refers to face-to-face communication and relates only to the function of referencing the transmission of information between two individuals [2]. This allows for only a few necessary functions to be fulfilled by the communication.

More recent definitions refer to communication acts that reference an external world of objects and events [17]. The simplest possible communication act involves one person (A) transmitting information to another person (B) about some object (X) (see figure 1). The meaning of the conversation consists of "report" and "content" functions, and "command" and "relationship" functions. Thus, a successful communication act consists of managing or manipulating content and relationships in order to come to some agreement about object X.

Traditional man/computer interfaces use the metaphor of communication in a more restrictive manner. The computer is seen as a passive partner engaged in information exchange with a user who gives orders or makes requests while the computer does the work (see figure 2). Most computer users are not interested in communicating with computers, but rather want to use the computer to solve problems or accomplish a task. This model can be seen most readily in computer applications such as querying a

database, using a word processor, or manipulating a spreadsheet. Even when the user is given a more direct interface, he is still engaged in information exchange. In an information exchange activity, the person does not want to come to an agreement with the computer. The person wants the computer to retrieve object X.

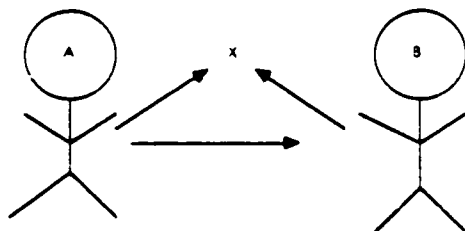


Figure 1. Person/Person

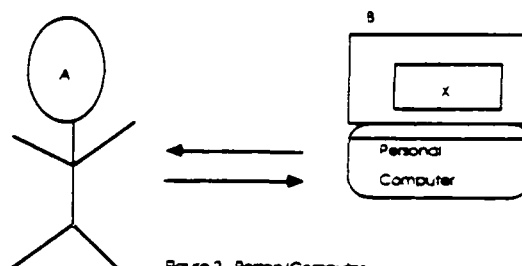


Figure 2. Person/Computer

More traditional computer assisted instructional (CAI) environments provide yet another metaphore of communication. In a CAI environment the computer controls the communication act by soliciting information from the student in the form of questions and requests (figure 3). Information exchange is again the purpose of the communication, but this time the computer is soliciting the information. Sometimes the computer even evaluates the student with respect to the quality of the information exchange. In this particular communication environment, the student is asked to retrieve information about object X.

Intelligent tutoring systems, more specifically student-centered intelligent tutors, try to model different levels of communication and adapt to the choices of the human user. Some of these systems [10] use the metaphore of iconic object manipulation to manage the interaction. This metaphore suggests that the student view items on the screen as objects which he can hold in his hand. Instead of referring to an object by its name, the student can use a mouse to point to an object or grab it and drag it across the screen. Some student-centered environments even allow the student to enter natural language expressions that communicate more naturally with the system [4]. In a student-centered environment, the computer and the student gradually become partners and learn how to manipulate content and relationships in order to come to an agreement about the knowledge they both share (see figure 4). I suggest that this type of communication management system is most similar to the face-to-face communication that is represented in figure 1, and is the type of communication that is required to make intelligent tutors more effective.

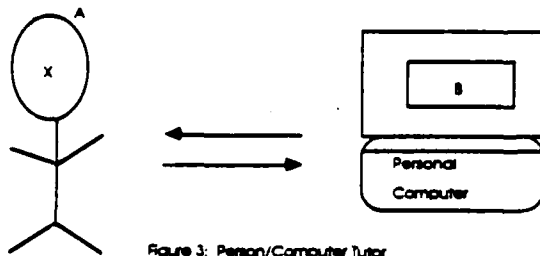


Figure 3: Person/Computer Tutor

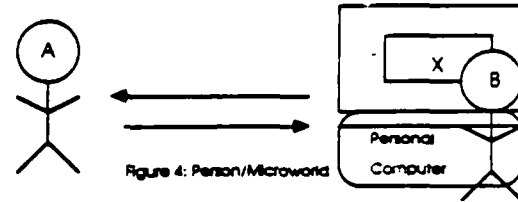


Figure 4: Person/Microworld

COMMUNICATION KNOWLEDGE IN INTELLIGENT SYSTEMS

There are two problems that must be solved by a designer of an intelligent tutoring system: 1) How to define the knowledge that the system is suppose to teach, e.g., teaching people how to land an aircraft, teaching people about a physics, etc., and 2) how to design a system that manages the interaction or relationship between student and computer so that the primary task can be achieved more quickly. Thus, communication knowledge in an intelligent tutoring system consists of rules and facts that tell the system how to manage the student/computer interaction.

Fischer and Morch [9] describe three approaches to communication styles used in human-computer interaction systems: tutoring, consultation, and critiquing.

Tutoring is important in the initial stages of training because a student needs to know something before he can ask a question. A tutoring system might be loosely defined as the arrangement of instructional sequences that eventually leads to the mastery of a specific goal. Thus, the designer of a tutoring system is responsible for arranging the sequences of instruction in a manner that facilitates the mastery of a specific objective. The communication management system, in turn, supports the instructional sequence and provides an interface that guides the student toward the goal. Thus, the major issue involved in constructing this type of interface is the question of limiting the users' choices to a finite number of selections. Although tutoring systems appear to be tolerant of students' responses, they impose a structure that can only react to anticipated responses. Examples of tutoring approaches include PROUST [20], the LISP-TUTOR [1], and SOPHIE [4].

Consultation is an interaction style that is used more frequently in expert systems than training systems. This particular model provides little support for learner controlled functions. The computer controls the dialogue by asking the user to enter responses to specific questions (much like what occurs when a student responds to a human consultant). Therefore the system is responsible for managing an interface that facilitates information exchange. At the end of the consultation, the

student receives an answer to his question. The student can ask the system why it gave a particular answer, but he cannot volunteer additional information nor can he pursue a particular line of reasoning. Examples of such systems include MYCIN [5], and an early version of GUIDON [7].

The critiquing model is used in more student-centered environment that permits students to pursue their own goals. The computer interrupts only when the student fails to meet minimum criteria. Based on previous research [6], it is known that students learn only what is necessary to solve the current problem. Thus, the critic intervenes only when it is necessary and only with advice that is problem specific. The critiquing approach provides information to the student only when it becomes relevant. The critiquing approach also allows the student to fail and make mistakes. Interface management for this type of model is much more difficult since the system cannot anticipate every action the student takes. Instead, the designer must provide a flexible environment that allows the student to explore different paths of action.

The author has developed a number of training systems that use different aspects of the critiquing approach. In order to help students better understand the relationships between orbital elements and ground tracks, we built an elaborate system that allows students to explore the "microworld" of orbital mechanics [22]. The original training system was designed to teach students how to deduce orbital elements by looking at ground tracks. Ground tracks are two-dimensional displays that show the portion of the earth that a satellite covers as it circles the earth. The actual satellite path appears as a continuous line on a monitor. The ground track information is then used by Air Force crews to verify orbital paths of known satellites, to hypothesize about mission intent of unknown satellites, and to monitor space debris. The ground track, as well as the orbit itself, is a direct function of the orbital elements.

The orbital mechanics tutor, nicknamed OM, allows the student to explore the interactions between orbital elements and ground tracks. Students learn about ground tracks by changing orbital parameters, changing injection points, generating ground tracks, predicting ground tracks, etc. Additionally, there are several online tools that help students organize and systematize their information. These tools include: a two-dimensional display that helps students conceptualize the effect of orbital parameters, definition and example windows that relate facts, a history tool that allows students to overlay previous work, and a prediction window that allows students to state relationships between variables. This system is designed so that students learn about orbital mechanics by gathering data, generating and testing hypotheses, as well as forming generalizations about the

different orbital elements.

The communication management system for OM includes a model of how students explore a microworld environment. The system monitors the student's actions and determines whether he is demonstrating effective inquiry skills. This is accomplished by simply associating all the online tools with specific categories of inquiry skills. When there is a sufficient amount of evidence to indicate that the student needs help, then the system suggests alternative problem solving strategies.

This same approach was used to develop a tutoring system called S-TRAINER (Strategic Training Routes Architecture for an Intelligent Effective Reviewer) which is designed to assist in the debriefing of Air Force pilots who have completed a bombing mission [23]. The system has the ability to reason about tactical situations and provide plausible explanations about these activities. Pre-stored mission events serve as script templates that are matched against actual events and the time relation between events. After the air crews complete the training mission, the system diagnoses pilot and crew errors, generates a written evaluation of the mission along with a set of graphics that can be used to supplement the written report.

S-TRAINER contains an elaborate communication management system that determines significant mission events and uses this information to generate a training script. The training script consists of a list of significant events along with a list of graphic displays that describe these events. All of the graphics are eventually shown on one of two screens. Each of the two screens, in turn, can be partitioned into a full, half, or quarter display. Thus the communication management system must decide on the content, type, and location of each display. In order to accomplish this task, S-TRAINER uses a series of rules that 1) suggest possible displays, 2) compute display times for each event, 3) determine whether to expand or reduce the number of displays, and 4) determine the correct flow of the script. Furthermore, the communication management system is responsible for producing a pedagogically sound and aesthetically pleasing training session. After several iterations, the system produces a final training script that indicates the type, length, location, and begin/end times for each display.

From the above discussion, it is clear that communication knowledge consists of rules that tell the system how to communicate task-relevant information to the student. For time-varying, multi-task problems, like teaching someone how to operate a space shuttle, the designer needs to consider several additional factors. For example, a format for an individual display must not only provide information about a specific task,

it must also blend with other displays that are presented simultaneously on the screen. It is becoming increasingly common for a format to be a member of a larger set, which is stored in software and presented on demand in different display areas. Consequently, the designer must make some provision for a smooth transition between formats as the different instructional events unfold. A poorly designed format will either fail to communicate critical information or demand more processing time than the student has available. The problem of constraining the design of a multi-media training system that uses simultaneous displays in a learner-controlled environment is the subject of the next section of this paper.

DESIGNING COMMUNICATION KNOWLEDGE

The creation of a student-centered communication management system requires special skills, special system capabilities, and special tools. The task is further complicated by the introduction of multi-media, multi-interactive, and simultaneous displays. A third problem relates to the human communication barriers that exist among those responsible for creating the training system - the psychologist, the computer programmer, and the instructional designer. Although the problem may appear to be overwhelming, the rather obvious solution is to develop a tool that will organize and systematize the development of a complex training systems.

Traditional computer science research has long suggested the importance of using software engineering tools to significantly improve the structure of programs and assist programmers with their problem solving tasks. Researchers argue that a design methodology can provide a focus and structure for solving computer problems [3]. A design methodology is an artificial language that enables the programmer (and any other member of the design team) to describe a particular idea at a conceptual, rather than implementation level. A design methodology also encourages a consistent design that can be shared among all the members of the team including the psychologist, instructional designer, and the programmer.

The problem, of course, is selecting a design tool that properly describes the computer's actions and reactions to a student's inputs. In short, the design tool needs to represent a communication process that often appears to be non-deterministic.

One way to distinguish between the various design methodologies is to classify them according to the declarative and procedural paradigms. Procedural representations are used to describe knowledge about how to perform a task. In contrast, declarative systems are used to represent descriptive

information. In some ways, the tension between the declarative and procedural representations has made it difficult to propose a design methodology for intelligent tutoring systems. Most designers of ITS systems believe that both the storage and retrieval of expert knowledge is best facilitated by the frame/object orientation [7]. Yet, when researchers wish to describe how their systems actually work, they use a procedural approach to explain the student/computer interaction [7]. A procedural representation more closely resembles the communication act. A procedural representation can best describe how one process activation is often dependent on the outcome of another.

The majority of the languages used to specify human-computer interfaces emphasize procedural information. Although these procedural methodologies go by different names (state transition diagrams [15], Petri Nets [11], and augmented data flow diagrams [13]), they all present a formal design model that looks very similar to a directed graph. In every case, nodes are used to represent the completion of events, whereas arcs represent transitions from one event to another. Different system responses are shown on the arcs and may involve an invocation of another part of the graph. Thus, the designer can represent both the conceptual and detailed structure of the student/computer interaction. In addition, a designer can represent a student's exploration of a single state by drawing an arrow from a node to itself.

An example of the use of a procedural methodology to describe the Orbital Mechanics student/computer interaction is presented in figure 5. As previously mentioned, the Orbital Mechanics tutor allows students to discover relationships between orbital elements and ground tracks. The student initiates a discovery activity by changing one or more orbital parameters and generating a ground track. After investigating the effects of changing different parameter values for different ground tracks, the student can advance to the Prediction window where he can make a hypothesis regarding the shape of a particular ground track. The student tests his predictive powers by selecting options from the menu and comparing the inputs to the Expert's conclusions. After making several successful predictions, the student enters the Orbit Prediction environment which is designed to check the student's mental state by asking him to perform a task in the reverse order of the one previously described. The student is shown a specific ground track and asked to enter orbital descriptors that match the ground track displayed on the screen. In this manner, the student explores the orbital mechanics microworld. Figure 5 describes this interaction and shows how the student advances through the program.

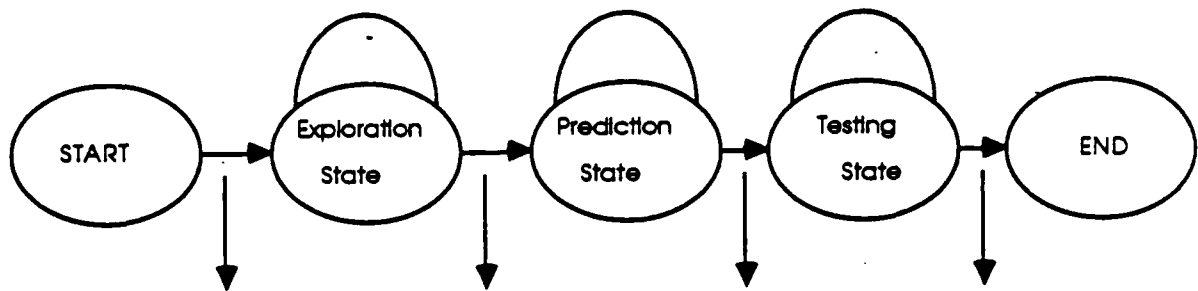


Figure 5: STUDENT/COMPUTER INTERACTION FOR OM

There are many compelling reasons for using the above methodology to construct student-centered environments. There are also a number of reasons why another design tool might be more appropriate. It is obvious, for example, that flow diagrams are cumbersome for specifying highly interactive environments. Higher-level, more directly-executable specification languages, would free the designer of the low level details. Yet it is also obvious that a design methodology helps support knowledge abstraction and computational primitives at the architectural level. Such a methodology permits the knowledge engineer to cooperatively develop systems using a shared language of conceptual constructs, rather than a set of problem specific primitives. A design methodology for a learner controlled environment is especially important since this form of problem solving is considered more complex than other types of tutors.

EVALUATING COMMUNICATION KNOWLEDGE

Although a common methodology allows us to better describe communication knowledge, there remains the problem of determining whether the communication is effective. A review of the literature on student-centered environments reveals only a minimal number of studies that contain hard data on the effectiveness of their instruction [12; 19]. Although many current AI systems emphasize the interactive and exploratory nature of learning, it is not clear that systems that use these principles are effective [18]. The evaluation of intelligent tutoring systems has always been predicated on behavioral experiments that determine whether one treatment is better than another treatment. The typical methodology is to construct a system, and then to use the system with multiple subjects and multiple trials to obtain reliable measures on the effectiveness of a treatment. These experiments are usually difficult, costly, and time consuming to conduct. These experiments become even more difficult to perform as researchers develop training systems that use a variety of different media and interactive styles.

A few evaluation studies attempt to discover which features of a tutoring system affect individual users [19]. It is

expected that this information can be used to predict the behavior of individual users engaged in complex problem solving tasks. The factors that affect user behavior are quite complex and interact extensively with one another. Thus, information that predicts how an individual student reacts to an exploratory environment should help us produce more effective training systems.

Another approach to the evaluation issue is to determine patterns of behaviors that characterize different groups of student interactions. For example, two groups of students enrolled in the Undergraduate Space Training School at Lowry Air Force Base, were asked to use the Orbital Mechanics tutors. Subjects were classified as either beginners (students in their first week of the program) or experts (students in their last week of the program). Subjects belonging to the two different skills levels were told how to use the system and then asked to interact with the system in a self-paced manner. A data collection program recorded all student/computer interactions. The data collection phase began after the students entered their first input and continued for, no more, than 15 minutes. Twenty subjects served in this experiment: 12 experts and 8 beginners.

An analysis of the subjects' interactions was performed using the Reitman and Rueter algorithm [16] which clusters individual responses into groups of responses that always appear contiguously, regardless of order. This particular analysis showed that expert subjects demonstrate a greater variety and depth of interaction. The expert students used more online tools and used them in a more consistent manner. Further analysis suggests that, as a group, the expert subjects were more similar to each other than they were to their beginner counterparts. The expert subjects tended to pursue goals in a very planned and prescribed manner [24]. The beginner students showed no similarity to each other, and tended not to pursue any clear goals.

What this particular study shows is that there is a correlation between expertise and knowledge organizations. This idea has already been confirmed by other studies on expertise [8; 14]. What this study also suggests is that we can use this type of information to identify different patterns of skills which should, in turn, help us to build better communication management systems. Pattern matching is one of the standard themes that appear throughout the artificial intelligence literature, and this study suggests that we should continue to develop this theme.

RESEARCH OPPORTUNITIES IN THE AREA OF COMMUNICATION KNOWLEDGE

A number of near-term and long-term research opportunities seem obvious at this point. A few of these goals are listed below.

Near-Term Goals

- * We need to validate a methodology for developing communication knowledge. The methodology might be similar to the one proposed in this paper, or it might include other types of design tools. The methodologies should examine how to better represent communication knowledge.

- * We need to develop a nationwide database that classifies existing media according to its content and cognitive function. This country, as well as other nations, have produced a large number of films and video disks. A simple database would allow us to extract various media selections in order to enhance existing instruction.

- * We need to expand our definition of student-centered environments to include interfaces that facilitate computer-supported cooperative environments. Cooperative problem solving refers to the shared, systematic behaviors displayed by two or more people as they work towards the solution of a single problem. As we begin to explore the issue of cooperative work environments [21], we might also ask whether people know how to use these cooperative work environments for group decision making. If this is not the case, then we need to develop systems that teach people how to become more effective problem solvers in a computer-supported environment.

Long-Term Goals

- * Interfaces should adapt to their users. Some of the advances in machine learning suggest that we can build interfaces that bridge the gap between the user's model of the information and the system's organization of the data. This initial success indicates that we can use these types of techniques to alter the interfaces and accommodate different learning styles.

- * Design methodologies for intelligent tutors should generate the code for intelligent tutoring systems. Many computer aided software engineering (CASE) products are already generating Cobol, C, and Ada code. [13] suggests that a tool already exists that can generate three different styles of user interfaces. It seems likely that a such a tool could be used to generate interfaces for student-centered environments.

* There should be a communication management science that would enable us to predict a student's interaction style. Such a science may soon be available as researchers examine cognitive functions and the effects of attention on dual task activities [26]. Such information is crucial to our understanding of how students learn and understand information.

REFERENCES

- [1] Anderson, J., and Reiser, B., The LISP Tutor, Byte, 10, April, 1985, pp. 159-175.
- [2] Bateson, G., Information and codification: A philosophical approach, In Communication: The Social Matrix of Psychiatry. J. Ruesch and G. Bateson, Eds., W.W. Norton, New York, 1951, pp. 168-211.
- [3] Boehm, B., Software Engineering Economics, Prentice-Hall, Englewood Cliffs, N. J., 1981.
- [4] Brown, J., and Burton, R. Pedagogical, natural language and knowledge engineering techniques in SOPHIE I, II, III. In Intelligent Tutoring Systems, D. Sleeman and J.S. Brown, Eds., Academic Press, New York, 1982, pp. 227-279.
- [5] Buchanan, B. and Shortliffe, E., Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project, Addison-Wesley Publ., Reading, MA, 1984.
- [6] Carroll, J. and McKendree, J., Interface design issues for advice-giving expert systems. Commun. of the ACM, 30, 1, 1987, pp. 14-31.
- [7] Clancey, W. Knowledge-Based Tutoring, MIT Press, Cambridge, Mass., 1987.
- [8] Engle, R. and Bukstel, L., Memory processes among bridge players of differing expertise. American Journal of Psychology, 91, 1978, pp. 673-689.
- [9] Fischer, G. and Morch, A., Crack: A critiquing approach to cooperative kitchen design. In Proceedings on Intelligent Tutoring Systems, Montreal, Canada, 1988, pp. 176-185.
- [10] Fischer, G., and Lemke, A., Construction kits and design environments: Steps toward human problem domain

- communication. Human-Computer Interaction, 3, 1988, pp. 105-160.
- [11] Jentzen, M., Structured representation of knowledge by Petri nets as an aid for teaching and research. Lecture Notes in Computer Science, Springer-Verlag, Berlin, 1980.
 - [12] Hoecker, D. and Elias G., User evaluation of the LISP intelligent tutoring system. In Proceedings of the Human Factors Society, Dayton, Ohio, 1986, pp. 182-185.
 - [13] Kuo, F., and Karimi, J., User interface design from a real time perspective. Commun. ACM, 31, 12, 1988, pp. 1456-1473.
 - [14] Larkin, J., McDermott, J., Simon, D. P. and Simon, H., Expert and novice performance in solving physics problems. Science, 208, 1980, pp. 1335-1342.
 - [15] Ling, M., Designing data entry programs using state diagrams as a common model. In Proceedings of the 6th International Conference on Software Engineering, Tokyo, Japan, 1982, pp. 296-308.
 - [16] McKeithen, K. Reitman, J., Rueter, H., and Hirtle, S., Knowledge organization and skill differences in computer programmers. Cognitive Psychology, 13, 1981, pp. 307-325.
 - [17] Newcomb, T., An Approach to the study of communicative acts. Psychological Review, 60, 1953, pp. 393-404.
 - [18] Sebrechts, M. and Deck, J., Techniques for acquiring computer procedures: Some restrictions on interaction. In Proceedings of the Human Factors Society, Dayton, Ohio, 1986, pp. 275-279.
 - [19] Shute, V., and Glaser, R., An intelligent tutoring system for exploring principles of economics. Tech. Rep. Learning Research and Development Center, Univ. of Pitts., Pitts., Penn., 1986.
 - [20] Soloway, E. M., and Johnson, W. L., Remembrance of blunders past: a retrospective on the development of PROUST. In Proceedings of the Sixth Cognitive Science Society Conference, Boulder, Colorado, 1984, pp. 57-60.
 - [21] Stefik, M., Foster, G., Bobrow, D., Kahn, K., Lanning, S., and Suchman, L., Beyond the chalkboard: Computer support for collaboration and problem solving in meetings. Commun. ACM, 30, 1, 1987, pp. 32-47.

- [22] Swigger, K., Burns, H., Loveland, H., and Jackson, T., An intelligent tutoring system for interpreting ground tracks. In Proceedings of AAAI-87, Seattle, Wash., 1987, pp. 72-76.
- [23] Swigger, K. and Holman, B. S-TRAINER: Script-based reasoning for mission assessment. (in press).
- [24] Swigger, K. An evaluation of the Orbital Mechanics tutor. (in press).
- [25] Wenger, E. Knowledge Communication Systems, Morgan Kaufmann, Inc., Los Altos, Calif., 1987.
- [26] Urbanczyk, A. A., Angel, C., and Kennelly, K., Hemispheric activation increases positive manifold for lateralized cognitive tasks: An extension of Stankov's hypothesis. Brain and Cognition, (in press).
- [27] Yazdani, M. Intelligent tutoring systems: An overview, In Artificial Intelligence and Education Vol. One, R. Lawler and M. Yazdani, Eds., Ablex Publishing, Norwood N. J., 1987.

FROM TRAINING TO TEACHING: TECHNIQUES FOR CASE-BASED ITS

Christopher K. Riesbeck
Roger C. Schank

The Institute for the Learning Sciences
Northwestern University
Evanston, IL 60201

1 INTRODUCTION

This chapter will focus on the teaching of knowledge-intensive domains, such as biology, history, and weather forecasting, in contrast to other chapters that discuss teaching high-performance skills (e.g., Fink and Regian) or basic concepts (e.g., Bonar, Porter and Woolf).

It has become a truism in artificial intelligence (AI) that knowledge is crucial. Clever algorithms, fancy data structures, and elegant formalisms won't save an AI program that doesn't know what it's doing. Success in AI depends on both quantity of knowledge (how much the program knows) and quality (how accurate is the knowledge, how accessible is it, and so on).

Consider what distinguishes a good encyclopedia from a bad one. It is not the "data structures" involved; both have labelled articles and a master index. Rather, it is the quality of the articles and the appropriateness of the index terms that matters. Similarly, an AI program—or a person—needs the right knowledge *and* the right indexes to be intelligent. It follows that a major task in education is to teach both knowledge and the indexes that make that knowledge accessible.

But that raises the classic AI question "how should knowledge be represented?" and the classic educational question "how should knowledge be communicated?" In much of what is called "knowledge-based AI," the representation question is answered with "rules." Rules have the form "IF situation THEN conclusion." Problems are solved and situations are understood by chaining rules together.

There are a number of problems with this approach. Coming up with a reasonable set of rules is a very difficult, if not impossible, task for many domains. Even in domains with rules, such as mathematics, the computational complexity of combining them to form inference chains may be too great to be feasible.

Furthermore, rules have their limits as a way of communicating knowledge. While detailed formulas and general principles are rule-like, neither are sufficient for describing a domain. Real domains, such as weather forecasting, have so many detailed formulas and so many exceptions to the general principles that a student would be overwhelmed by any tutoring system that tried to present them all.

There is an alternative to rules, however, and that is to represent domain knowledge in the form of many example cases that embody, either explicitly or implicitly, the relevant formulas and principles. Problems are solved by adapting previous solutions for similar problems. Situations are understood by adapting previous interpretations for similar situations.

Case-based reasoning has a number of potential advantages for solving AI problems, but our focus in this chapter is on the impact this technology can have on intelligent tutoring systems.

2 CASE-BASED TEACHING

It is well understood in many complex domains that the best teaching method is case-based. Law schools and business schools teach cases rather than rules. Recent research in AI suggests that there is a valid psychological reason for this, and we are now designing AI teaching methods that use cases to teach. We believe that many, if not most, skills can be taught in a case-based fashion.

Most training involves teaching a trainee to know what to do and when to do it. A trainee is successful to the extent that he does what he is told to do when he is supposed to. The trick is that the trainee has to abstract from the situations he has been told about to ones for which he was not specifically trained.

To learn to deal with specific cases, one must learn the prototypical cases, the standards that serve as the basis from which one learns how to deal with a new situation. A computer system that is a real-life simulation can create a situation, posed as a problem to the trainee, and ask for a response. A good teacher forces conjectures on the part of the trainee and simulates real world situations. A trainee would be expected to respond with the right answer, or, failing that, would be presented with a situation where that question has been asked before and where a good answer was given in response. The trainee would then be encouraged to try a variation of the previous good answer. Learning by copying is an important part of learning. Learning to move from copying to creative adaptation is another. And learning to know what cases to try to adapt, that is, learning what makes cases relevant, may be the most important part of all. Thus, in our approach, a trainee is developing a battery of situations—cases—and appropriate ways to respond in those situations.

A recent book, *Thinking in Time*, by Neustadt and May, describes how decision making occurs in government and how the authors actually go about teaching decision makers to improve their ability in this area. A great deal of what is said in the book indicates that government decision makers, like everybody else, reason by reminding. They need to make decisions, so they recall similar situations and try to reason from them.

Two situations in the book are noteworthy in this context. The first is the Bay of Pigs invasion and the second is the *Mayaguez* incident. Decision makers in both cases recalled and argued from past cases in history. In the Bay of Pigs case, the prototype was an incident in Latin America ten years earlier that caused a little-known invasion by the U.S. which seemed to work. In the *Mayaguez* incident, the prior case was the *Pueblo* incident from a few years before.

In both of these reminders, we have a situation where decision makers had an incident brought to mind that had occurred a few years before, that is, within the memory of decision makers, and that was *superficially* similar. I emphasize the word "superficial" because, as the authors point out, that's all that was in common. The *Pueblo* and the *Mayaguez* were both ships with Spanish names flying U.S. flags captured in Asia, but one was a military ship while the other was commercial. One was captured by an enemy government, while the other was not. In fact, these incidents had very little in common with the exception that when President Ford heard about the *Mayaguez* decided not to do what had been done with the *Pueblo* and almost created a great deal of difficulty for the people he was trying to save.

The problem here is that while reminding is the right and natural thing to do, it doesn't mean much if you don't have much to be reminded of. There is a real problem with the corporate memory of the government and of the world in general. Education means having enough cases available so that when a decision needs to be made, one or more very close matches from history come to mind. The more relevant evidence available, the more an intelligent decision is like to result.

Thus, it becomes clear that a system of selecting and presenting cases to a decision maker can be an important means of enhancing good decisions. For a student, interacting with a system that can present past history when needed allows for the possibility of learning what one needs to know when one wants to know it. Teach a student about Viet Nam and he may fall asleep. Charge him with having to make a decision about Nicaragua and he may, all of a sudden, become quite interested in Viet Nam. If a student suggests invasion, a suitable historical precedent is retrieved and described. If he mentions economic support, a different prior story would be told. If he mentions ignoring the situation, yet another story surfaces, and so on.

Case-based teachers, then, would be well-indexed libraries of prior situations. The trick in creating them is indexing them properly. Good teachers try to present facts and alternative interpretations as neutrally as possible. This is quite difficult for people, but, for obvious reasons, much easier for machines.

Most of all, such systems would allow students to do hypothetical reasoning in complex domains while becoming active learners about subjects they themselves decide to explore. This is what education can and ought to be about. Since learning is really the accumulation of cases, learning and creativity involves the adaptation of imperfectly fitting cases to new situations.

3 CASE-BASED REASONING

The most common model of cognition in AI is *rule-based reasoning*. It distinguishes two kinds of knowledge, facts and procedures. Facts are represented using propositions, frames and/or semantic networks. Procedures are represented with IF-THEN inference rules. To explain a situation or solve a problem, a rule-based reasoner applies various combinations of rules to various facts until a chain of reasoning, i.e., a proof, is found. The proof may be wrong, because the rules and facts are dubious and the combination rules heuristic, but a rule-based reasoner remains a very *thoughtful* creature. Usually, however, a rule-based reasoner is also a very *forgetful* creature. Proofs are forgotten, so that when the same or similar situation arises again, the same problem-solving behavior must be reenacted.

We have argued elsewhere, however, that everyday intelligence is just not that thoughtful and certainly not that forgetful [Riesbeck and Schank, 1989]. People quickly relate events and problems to prior experience. When personal knowledge is stretched too far, as when the man in the street is asked to evaluate complex political events, absurd responses can result. But in familiar situations, sophisticated answers come quickly and robustly, as when the average person goes to a restaurant or shops for groceries, an experienced car mechanic sees a familiar situation, or a chef encounters a tricky situation he or she has dealt with before.

We call reasoning from experience *case-based reasoning* (CBR) [Schank, 1982; Kolodner and Riesbeck, 1986; Kolodner, 1988]. CBR is important in domains such as politics or cooking, because it is very difficult, if not impossible, to formalize them with rules. In these domains, reasoning from examples is the only serious option. CBR is also important in domains that have too many rules, as in weather forecasting or economics, or too many ways in which the rules can be applied, as in mathematics, programming, or game playing. In these domains, cases suggest approximate answers, thereby limiting how many rule combinations must be explored.

At the heart of CBR are two processes: *indexing* or labelling, and *adaptation*. When new experiences come into the system, they need to be labelled for future retrieval. When retrieved to deal with a new situation, cases needed to be adapted to fit the current circumstances.

Cases need two kinds of indexes, concrete and abstract. Concrete indexes refer to objects and actions usually directly mentioned in the case. Abstract indexes refer to more general characterizations of the case. For example, a chef might index a recipe by concrete indexes such as ingredients and basic type (stew, stir fry, casserole, etc.), as well as by more general characterizations, such as how easy it is to prepare, what kind of people like it, what kind of cooking problems it solves, and so on.

The *indexing problem*, as it is usually called in CBR research, is the problem of determining the appropriate abstract and concrete indexes for cases. How we index incoming cases (instances of going to a restaurant, proofs by contradiction, monetarist arguments, and so on) determines what cases we will compare the inputs against. A very general index will cause a case to be retrieved even when it shares no specific details with the current situation. *Romeo and Juliet* is useful for understanding *West Side Story*, even though the details differ greatly.

Figure 1 shows the reasoning and learning components of a typical CBR system. An input describing a problem or situation is analyzed and used to retrieve one or more similar cases in the case library. The solution that was used in the most similar case is then adapted to solve the input problem. If it works, the new solution is added to the case library. This is *success-driven learning*. Ideally, a CBR system should never repeat the same problem-solving process. If the new solution fails, then two things happen. First, the failure is analyzed and the solution repaired. If the repaired solution works it is added to the case library. Second, new indexes for labelling cases are created, based on the failure analysis, so that similar inputs in the future will retrieve the repaired solution, not the one that led to the failure. This is called *failure-driven learning*. Ideally, a CBR system should never repeat its mistakes. More details of case-based reasoning, along with examples 10 of various CBR systems can be found in [Kolodner and Riesbeck, 1986; Riesbeck and Schank, 1989].

Case-based reasoning relates to ITS research in two ways. First, CBR is a *model* of cognition and learning that suggests that the goal of an ITS system should be to teach cases and how to index them. Rules can be useful, even crucial, but they don't become intuitive and useful until there is a rich case base showing when the rules apply and when they don't.

Second, CBR is a *technology* for building ITS systems, a technology that stresses the construction, indexing, and use of large libraries of related, sometimes redundant, examples, rather than the development of inference engines and highly-tuned sets of rules.

In the sections that follow, we are going to describe three complementary technologies for building case-based intelligent tutoring systems:

- case-based reasoning itself, in particular the indexing of cases with labels that allow cases to be used in a variety of circumstances
- direct memory access parsing, to enable natural language access to cases
- reactive tutorial plans that take into account the contents of the case base, current pedagogical goals, and the state of the student

4 INDEXING CASES

If we simply indexed cases with words and phrases:

- synonyms would have to be included for each index word;

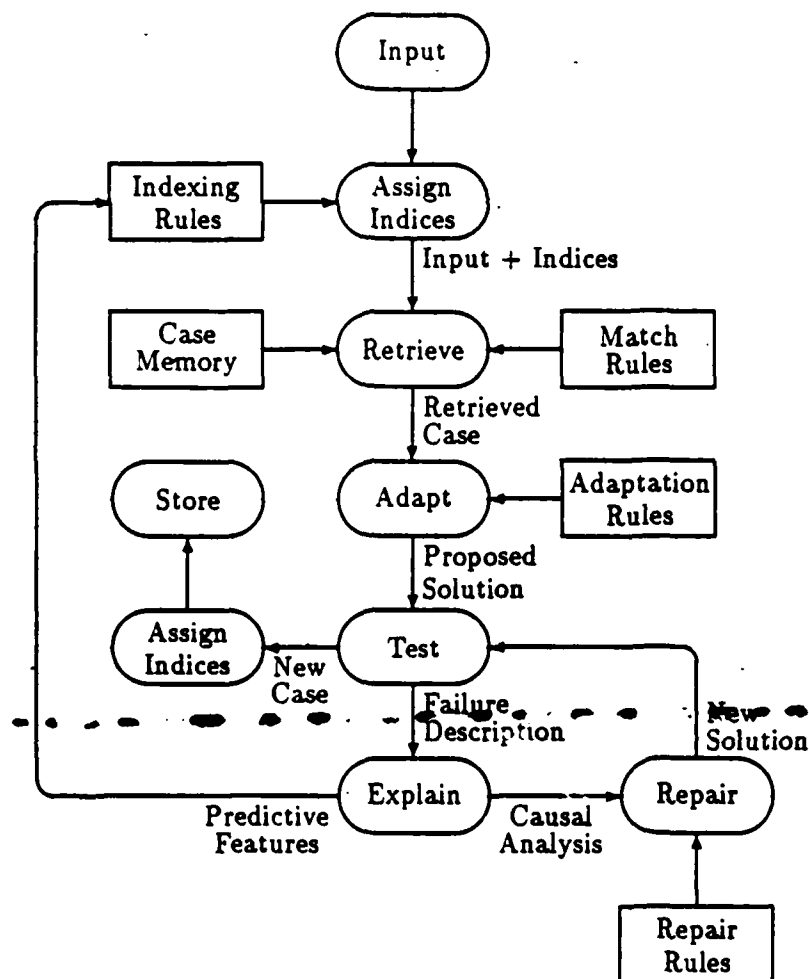


Figure 1. CBR Flow Chart.

- cases would have to be reindexed when the language of the student population changed;
- common uses of words might not agree with the case library's, e.g., "bug" and "insect" are synonyms to the average person, but not to a case library about entomology;
- ambiguous words would retrieve unrelated cases, e.g., "bug" would index cases in entomology and computer programming; and
- the student wouldn't know why a case was retrieved, e.g., "beef" and "inflation" might both retrieve a description of Argentina, but for different reasons.

We need to separate lexical connections from conceptual ones. Linking words and phrases to concepts should be taken care of by the interface, as discussed in the language understanding section below. Furthermore, we need to indicate how an index concept relates to a case. Thus, we might index a case as "a kind of ...," "an instance of ...," "a part of ...," "the cause

of ...," and so on, where each "..." is filled in by some concept. Beyond these simple semantic relationships are others such as "is analogous to ...," "is often accompanied by ...," "is a prototypical instance of ...," "is an exception to ...," "is summarized by ...," and so on.

Consider an ITS in the domain of programming, and how powerful it could be if it knew that:

- Computer programs are analogous to cooking recipes.
- The bubble sort is a classic instance of a computer program.
- PROLOG is an exception to the procedural view of programming languages.

Even in simple semantic networks, these kinds of indexes can be useful: robins and sparrows are prototypical birds, the penguin and ostrich are exceptional, one goal of flocking behavior can be summarized by "safety in numbers," and so on.

Getting these indexes means analyzing domain experts, tutors, and students to find the concepts each uses to organize experience. The concepts students use need to be tied to the concepts domain experts use. Some of what a student knows can be characterized as instances or examples of specific domain concepts, e.g., *BASIC* (a student concept) is an instance of a *procedural language* (a domain expert concept). Other student concepts can be characterized as abstractions of more specific domain concepts, e.g., *monkey* (student concept) is really a subclass (domain expert concept) that excludes chimpanzees.

5. CONNECTING LANGUAGE TO KNOWLEDGE

Teaching knowledge-intensive domains such as biology or history commonly involves natural language. There are texts, summaries, questions, answers, problem specifications, and so on. Even in weather forecasting, where problem situations are specified with maps and numeric annotations, there are the briefings where the forecaster summarizes his or her analysis and prognosis, and the instructors critique the briefing. Certainly, our normal picture of a tutor-student interaction involves natural language dialogue.

The problem is that natural language understanding is one of the hardest problems in AI. There are no systems that can *understand* a significant corpus of text; that is, there are no systems that can produce meaning representations usable by a reasoning system or an ITS. The difficulties are particularly noticeable in dialogues, where grammatical rules become the most context-dependent. The classic example of this is the sentence "George thinks vanilla." It looks non-grammatical and nonsensical in isolation, but is in fact perfectly fine after the question "What flavor ice cream does Mary like?"

Our approach has been to radically re-think what language understanding is all about. The standard model of language understanding is the *meaning construction model*: text is understood by putting pieces of meaning together. To understand "George thinks vanilla," fit together the pieces a *person named George*, the *action of thinking*, and a *flavor called vanilla*. The meaning construction model fails in this situation, because *vanilla* can't be an object or style of thinking. To fix this, the system assumes that vanilla is an elliptical reference, and searches memory for what it might be referring to. The system "changes modes," so to speak, going from construction to memory search.

In our view, language understanding is always memory search. Texts are always references to existing knowledge structures, and the goal of the language understander is to chase down

those references. The Direct Memory Access Parser (DMAP) [Riesbeck and Martin, 1986; Riesbeck and Schank, 1989] is an implementation of an understander based on memory search. When DMAP sees words and phrases, it searches for the concepts those words could refer to. When DMAP sees a sequence of concepts, it searches for larger memory structures that such a sequence could refer to, and so on.

Unlike most systems, DMAP does not have a lexicon of word senses. Instead, attached to each concept or case are patterns or sequences of words and concepts that might be used to refer to that concept or case. By intersecting references as the text is read, DMAP determines which memory structures account for the text as a whole. (DMAP uses a marker passing algorithm, described in detail in [Riesbeck and Martin, 1986; Riesbeck and Schank, 1989].) Figure 2 shows the reference sequence that recognizes "George thinks ...," which is attached to the believe memory structure.

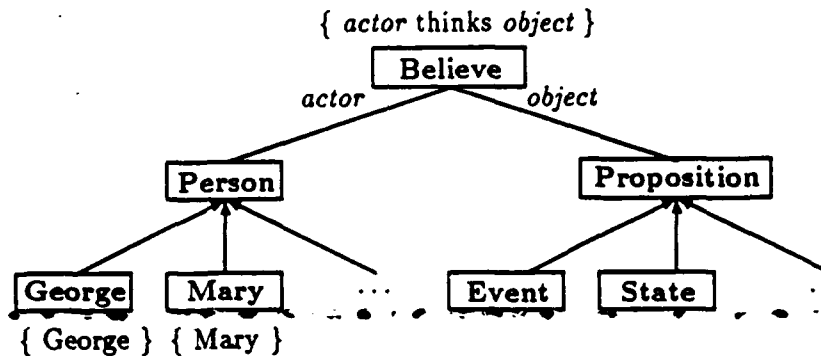


Figure 2. DMAP Reference Sequences.

DMAP has memory structures representing dialogue structures, such as questions and answers, with reference sequences attached to different kinds of questions, e.g., "What *object* does *actor* *action*?" and answers, e.g., "*object*." One kind of answer is a version of the believe memory structure, where the *object* of the believing is the answer to the question. The question "What flavor ice cream does Mary like?" followed by the statement of belief "George thinks vanilla" satisfies the reference sequence for a particular question and answer memory structure.

We believe that the memory search approach is ideal for ITS for several reasons:

- Language entry is simpler, i.e., attach sequences to memory structures; getting the right memory structures remains hard, but an intelligent system needs them, whether or not language is involved;
- The reference model of language seems particularly appropriate to the highly referential nature of dialogues;
- Special cases are easy to capture, e.g., questions about certain topics or the use of particular phrases; and
- Only modest extensions is required to use reference sequences in reverse to describe concepts and cases in natural language; choosing what to say when remains a hard problem, but putting it into words is not.

6 TUTORIAL PLANS

A third problem in teaching knowledge-intensive domains is covering the curriculum. The more the student discovers things on his or her own, the better, but an ITS should take advantage of any situation where important topics can be introduced. For example, consider the following dialogue that took place between a human biology professor and a student, where the student was asked to design an animal.

STUDENT: I want to design a cow-like thing. But I want my cow to have six legs.

PROFESSOR: I'll let you if you want. But I'll tell you that there aren't any.

STUDENT: Why aren't there?

PROFESSOR: That never happened because cows evolved from animals that only had four legs. Not that it's impossible, but that it never happened. That's an interesting point: several things we could get to are possible, but they just never happened. That's the way history went.

STUDENT: I'm going to give it six legs.

PROFESSOR: Go ahead. What are the legs for?

STUDENT: I want it to run fast.

PROFESSOR: Why do you think six legs will make it go fast?

STUDENT: You tell me.

PROFESSOR: I don't think you can reason whether it will or not because there are six-legged things around. Some of them are very fast for their size, like ants. Insects in general really truck out.

The types of the answers the professor gave depended very heavily on exactly what he knew and what he thought was interesting to teach. For example, when the student mentioned a six-legged cow, the professor knew that no such thing existed, that no analogous creature existed, such as any six-legged mammal, and that the reason was more evolutionary than functional. If the professor knew about six-legged cows in Borneo, he could have mentioned them. If he knew about six-legged cats, he could have mentioned them. If he knew that mammals can't function with six legs, he could explain why. In this case, the professor talked about evolution because (1) it could be related to why cows don't have six legs, and (2) evolution is an important concept in biology.

Although the student asked the questions and proposed the variations, the content and options made available in the dialogue were controlled by the professor, who opportunistically inserted examples, generalizations, questions, and so on, as he felt appropriate. This is called a mixed-initiative dialogue [Carbonell, 1970]. One problem with achieving mixed-initiative dialogues is that it is hard to specify in advance a set of tutorial scripts or plans that can apply in all circumstances. Whether or not it is appropriate to answer a student's question with another question, or an example, or a hint, depends on the content of the question, the expertise of the student, the history of the dialogue, and so on.

Until recently, plans in AI were very rigid. For example, a plan for getting milk might be "get in car; drive to store; buy milk; drive home." There was a sharp division between *planning*

time when the plan was constructed, and *execution time* when the plan was executed [Charniak and McDermott, 1985]. Such a model of planning can't work in the real world, where too many things are unknown to be able to specify all actions completely. Where is my car exactly? On which side of the garage? Does it have enough gas? Has construction finished on Main Street? Has the price of milk changed? Do I have enough money in my wallet?

[Firby, 1989] redefines plans to consist of sets of *reactive action packages* (RAPs) rather than sequences of actions. Each RAP is a little program responsible for achieving some small goals, such as "get in car." A RAP tests to see if its goal is already achieved. If not, the RAP selects an appropriate method to execute to achieve that goal, based on the current situation. The method might be a primitive action or another RAP. Thus, my "get in car" RAP would have a test "be in car," and methods for achieving that goal, such as "look to see where car is; go to that location; get inside car." "Get inside car" would itself be a RAP that would unlock the car if necessary, open the door, and so on. A RAP monitor keeps the RAPs in a queue and selects for execution those RAPs with important goals or impending deadlines. The key point is that many low-level planning decisions are deferred until execution time, allowing the system to deal with unexpected obstacles and take advantage of unexpected opportunities.

This execution-time context-sensitivity is the key feature we want to exploit with our tutorial RAPs. The goals will be pedagogical: concepts to be covered, skills to be confirmed, difficulties to be resolved, and so on. The methods will be dialogue techniques: give a hint, give an analogy, ask a leading question, tell a story, review a previous case, and so on. The tests will look not at the real world, but at memory: past elements of the dialogue, prior experience with the student, examples with a given set of features, and so on. For example, a tutorial RAP might look for examples of non-social insects in memory. We see our tutorial RAPs as being much like the DACTN's described by Woolf in this volume, with two differences. First, we are most interested in RAPs that make opportunistic use of domain knowledge found in memory, and, second, we want to encode our tutorial RAPs as memory structures.

7 A CASE STUDY

It's easy to say that a intelligent tutoring system should have such and such properties. Building such a system is another matter. Calls for mixed-initiative dialogues, for example, are almost as old as the field of computer-based education. The problem is getting from where we are to where we want to be. Just as it's hard for students to learn in isolation from real-world contexts, so it's hard for ITS designers to construct solutions separate from some real system. But it's easy for an ITS design to become an all-or-none affair. Either domain knowledge and linguistic abilities are there, or nothing works.

Our approach is to build a tutoring system in phases, where the early phases build the knowledge base and the later phases add the intelligent interaction. A key constraint is that every phase should be a useful product in its own right, and that each phase is more attractive to the student than the previous phase.

As an example of what we mean, we'd like to describe a design we proposed for a case-based tutoring system in the domain of weather forecasting. This is a very preliminary design, inspired by conversations with personnel at the Weather Training Division at Chanute Air Force Base. The proposals and claims made below, however, are those of the authors alone.

7.1 Weather Forecaster Training

The Weather Training Division at Chanute Air Force Base in Rantoul, Illinois, trains weather forecasters for all of the armed services. The intensive 22-week course covers everything from

basic principles of weather and climate to detailed techniques for analysis and forecasting. The students are high school graduates, normally with several years of service, but with no significant experience in forecasting.

The course is a mixture of class and lab work, where lab work means applying techniques taught in the classes to real weather data. At several points in the course, there is a major evaluation period, where each student analyzes a set of data and/or forecasts and presents a briefing to the instructors, who then critique both the analyses and the briefing style.

The students have a *lot* to learn and a *lot* to do. An analysis and/or forecast can take six or more hours to work up. Not surprisingly, the students have trouble managing all the data and rules. Many of the instructors' critiques of student briefings focussed on failures of the students to tie together factors from several sources, e.g., to connect the motion of a particular system to an upper air jet stream. Often the students would have a correct analysis, but miss many corroborating details.

7.2 A Sequence of Tools

As noted by Pirolli in this volume, a key issue is the integration of educational tools into the existing educational system. Introducing a new tool into a crowded educational curriculum is made much harder if it takes time from the class day, has to be forced on the students, or requires additional training for the instructors. Therefore, we considered introducing case-based ITS into the weather school incrementally, via the following sequence of tools for the student:

- a "dumb" *homework helper* with forms and tools to make it easy to do forecasts on-line faster and more neatly than by hand;
- a "dumb" *case browser*, callable from the homework helper, for scanning a *case library* of weather situations and forecasts;
- a "smart" *case retriever*, callable from the homework helper, capable of retrieving weather situations similar to the current exercise; and finally
- a *case advisor*, callable from the homework helper, that can give hints, advice, stories, cautions, and so on, relevant to the current exercise.

Each succeeding application improves the previous one in an obvious way, but each application is a useful tool in its own right. Figure 3 shows the basic set of modules.

The homework helper is the critical "foot in the door." To make the homework helper attractive to students, it should have

- exercise data already on-line.
- drawing and calculation tools that make doing analyses and forecasts faster or more accurate than doing them by hand.
- the ability to print answers that can be handed in.

An obvious part of such a tool would be a contour chart drawing tool for drawing isobars. The contour tool screen would show a map with numerical data points, just like the paper charts the students start with currently. The student would point and click at points in sequence

to draw a line, just like following the dots. The tool would automatically generate smooth curves. For the student, the result would be a faster way to generate cleaner charts.

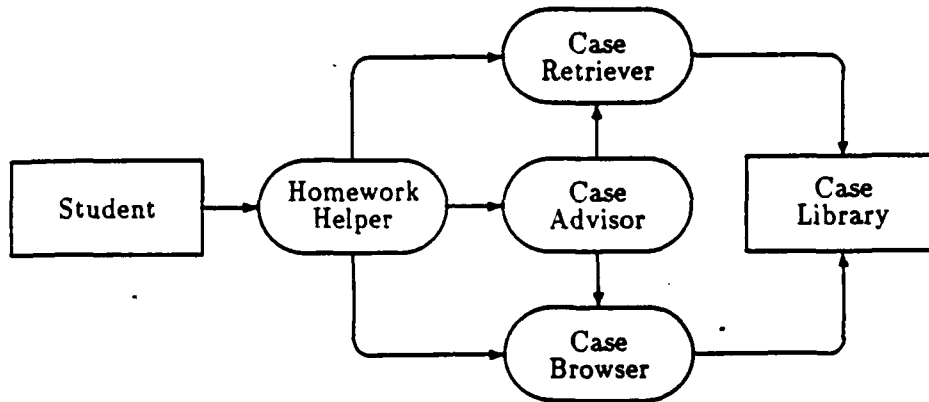


Figure 3. Forecast Tutoring System.

In addition, it becomes easier to check the charts for correctness. Currently, the students check their charts by laying a correct chart on top of theirs, and then laying on top of that a colored template with several holes that indicates key locations to compare. This is necessary because trying to match the entire chart would be too tedious. The student then has to decide if his or her lines are close enough. With the tool, the possibility exists for evaluating the charts automatically, comparing not the lines, but the data points selected for each line.

The case browser module would let the student or instructor specify weather features and retrieve weather cases with those features. The instructors could use this facility to find examples or exercises, and the students could use it to find situations similar to the problem they are working on.

We would like to use natural language as much as possible for browser requests, for reasons discussed earlier. Weather situations are very complex entities, however, not easily describable in one sentence. Therefore, we will probably design a "fill in the blanks" request form to help break the request down into well-defined parts. The blanks would then be filled in English.

The case retriever module introduces true CBR to the tutoring system. The retriever will find cases similar to the current exercise, using indexes encoded by domain experts. It will list both the cases retrieved and the index features that connect them to the current case. This list of features is important pedagogically because it captures how experts view weather situations.

The case advisor is the intelligent tutoring module, and the most complex and ambitious component of the system. The student will call on the advisor (using natural language) when he or she reaches an impasse during a problem solving session. Using pre-defined tutorial RAPs that opportunistically select different kinds of responses depending on what is in memory, the advisor may answer Socratically with a question of its own, give a hint, draw an analogy, refer to the cases returned by the retriever, or browse for cases with particular features.

8 LONG-TERM GOALS AND VISION

In our vision, the ideal case-based tutor is a raconteur, a teller of stories, a fountain of examples and exceptions. It communicates in the natural language of the student, pushes the student to make conjectures and ask questions, and presents cases not as solutions to problems, but as examples of what has been done before. Ideally, the tutor is fun to interact with because of the stories and cases it tells, not because it has flashy sound and graphics, arcade game interludes, scoring and competitive ranking, or whatever. The student learns because the student is the dominant actor in the conversation, led into exploring new concepts and ideas because they answer questions the student has raised for him or herself, based on hints and allusions dropped by the tutor in conversation.

On the other hand, our ideal tutor is *not* a test giver or a question answerer. Tests serve two purposes: as a stick to force the student to learn, and as a means of evaluating what the student has learned. The kinds of objective tests that are usually administered, especially on computers, with their blanks to fill in and multiple choices to select, do a terrible job on both goals. They don't force students to learn, because many students just "study for the test," learning little, and retaining less. Nor do they evaluate what the student has learned, because such tests only measure simple question-answering skills, and fail to give any indication of a student's creativity (in fact they stifle it) or ability to apply what's been learned in real situations.

Our ideal tutor is not a question answerer, because, in our view, learning results when the student both asks the questions and answers them. The role of the tutor is to either (1) help the student who is having difficulty to organize his or her attack on the problem, by asking a few leading questions, making a simple analogy, or showing an example of a similar situation, or (2) push the student with the glib answer into exploring alternatives, by presenting counterexamples unusual situations, and so on. Except for minor clarifications and pointers, we believe the tutor should never terminate discussion and exploration with "the answer."

In doing this, of course, we need to avoid frustrating the student with evasive non-informative interactions. Part of the solution is to put the student in charge of the problem-solving or exploration, and make the tutor more of a Doctor Watson to the student's Sherlock Holmes. That is, the tutor should prompt the student to think things through, carry the tools the student needs, bring up issues that should be examined, and so on, but should not be viewed (by the student or the designers of the tutoring system) as the "brains of the outfit."

Our proposed weather tutor is certainly a long way from our ideal, but it does point in the right direction. It puts the student into a well-motivated context (making forecasts) with real examples to work on. As a homework helper, it does not have a lesson plan that says when a student has to learn something. Instead, it waits until the student wants to know something. As a knowledge resource to be called on when needed, it neither interferes with nor redirects a student's natural flow of problem solving. As a library of cases rather than solutions, it does not tell the student what to do, only what has been done. The role of the homework helper is clearly that of assistant, not holder of answers.

9 SUMMARY

We've described two aspects to the design of knowledgeable intelligent tutoring systems. First, we described the case-based reasoning (CBR) model of what learning a knowledge-rich domain involves. We also proposed the use of CBR for implementing the knowledge base of tutoring systems. Second, we described how a tutor should teach this knowledge, emphasizing guided exploration in a problem-solving context. A student would ask the tutor for help in solving a problem, and the tutor would help, albeit indirectly, with leading questions, hints, case histories, and so on. This dialogue would be guided by tutorial RAPs that are sensitive not

just to the history of the dialogue but also to what knowledge is in the tutor's case base and what pedagogical goals are active. The natural language understanding and generation would be handled by a knowledge-based technique called direct memory access parsing (DMAP).

Finally, we outlined an approach to the design of a homework helper for trainees in weather forecasting. We organized development of the helper into short and long term goals. In the short term, we are look to build useful tools, including on-line calculation and graphic tools for doing analysis and forecast exercises, a case library of weather situations, a "dumb" case browser, and a "smart" case retriever. As a long term goal, we would build a case advisor, with RAP-like memory structures for its tutorial plans, the weather case library as its source of examples, exercises, analogies, and so on, and DMAP-based natural language understanding and generation.

10 ACKNOWLEDGEMENTS

The research described above is funded by the Air Force Office of Scientific Research contracts AFOSR-87-0295. We are also very appreciative of the efforts of Lieutenant Colonel Hugh Burns and Major Jim Parlett at the Air Force Human Resources Laboratory, and Colonel Dan White and Major George Whicker at the Weather Training Division at Chanute Air Force Base, Illinois.

REFERENCES

- Carbonell, J.R. (1970). AI in CAI: An artificial intelligence approach to computer-assisted instruction. *IEEE Transactions on Man-Machine Systems*, 11(4), 190-202.
- Charniak, E., & McDermott, D. (1985). *Introduction to artificial intelligence*. Reading, MA: Addison-Wesley Publishing Co.
- Firby, R.J. (1989). *Adaptive execution in complex dynamic worlds*. Doctoral dissertation, Yale University, New Haven, CT.
- Fletcher, J.D. (1985). Intelligent instructional systems in training. In S.J. Andriole (Ed.), *Applications in artificial intelligence* (pp. 427-451). Princeton, NJ: Petrocelli Books, Inc.
- Kolodner, J.L. (Ed.). (1988). *Proceedings of the First Case-Based Reasoning Workshop*. Los Altos, CA: Morgan Kaufmann Publishers, Inc.
- Kolodner, J.L., & Riesbeck, C.K. (1986). *Experience, memory and reasoning*. Hillsdale, NJ: Lawrence Erlbaum Associates.
- Riesbeck, C.K. (1987). Parsing, expectation-driven. In S.C. Shapiro (Ed.), *Encyclopedia of artificial intelligence* (pp. 696-701). New York: John Wiley & Sons, Inc.
- Riesbeck, C.K., & Martin, C.E. (1986). Direct memory access parsing. In J.L. Kolodner & C.K. Riesbeck (Eds.), *Experience, memory and reasoning*. Hillsdale, NJ: Lawrence Erlbaum Associates.
- Riesbeck, C.K., & Schank, R.C. (1989). *Inside case-based reasoning*. Hillsdale, NJ: Lawrence Erlbaum Associates.
- Schank, R.C. (1982). *Dynamic memory: A theory of learning in computers and people*. Cambridge University Press.

ON THE ART OF BUILDING: PUTTING A NEW INSTRUCTIONAL DESIGN INTO PRACTICE

Peter Pirolli
Education in Mathematics, Science, and Technology
University of California, Berkeley

SUMMARY

Professional instructional designers are one set of potential consumers of the results of work in cognitive science and intelligent tutoring systems. One way of disseminating new theories of instructional design into design practice might be through the development of computer-based design tools. This paper provides a characterization of the design process and significant features of new instructional theories. It also presents an overview of two existing computer-based design systems. The first, ID Expert, is an expert consultation system which essentially guides the user through the design process. The second, IDE, is a hypermedia system that facilitates the structuring and manipulation of analyses and specifications by expert designers. Each system has its merits and drawbacks. Finally, new instructional design techniques are discussed that might be profitably incorporated into computer-based design systems.

INTRODUCTION

Work in the field of intelligent tutoring systems pursues a variety of goals. In part, the work is that of any descriptive science, attempting to understand human cognition and learning, and the instructional effects of particular technologies. In part, the field is an engineering discipline, involving the development of instructional principles and rationalized instructional designs. There are clearly a number of audiences or consumers for the results and artifacts produced by the field and one of these is surely the professional designers and builders of instruction. The main task of this paper is to examine how the field of intelligent tutoring systems, and more generally the field of cognitive science and education, may systematize and disseminate its theories and methods of design into professional design practice.

Lessons From History

It is interesting to reflect on the history of architecture, perhaps the most established of design disciplines, and consider a particular set of events that turned it into a systematic and well-defined profession, and produced the wide-spread dissemination of a certain style of design. Many historians [9, 25] regard Leon Battista Alberti's [1] *On the Art of Building* to be the first modern treatise on architecture. Alberti was a true Renaissance man in every sense of that word, being well-versed in a variety of humanities, arts, and sciences [6]. His work is remarkable because it is based on a reanalysis of ancient writings (in particular the work of the Roman engineer Vitruvius Pollio), the extraction of design principles from the classical Roman architecture that he observed throughout Italy, and the formulation of a theoretical basis for the practices of Brunelleschi and other contemporary builders at the beginnings of the Italian Renaissance. Alberti did more than collect a compendium of principles and practices: He developed a theoretical framework that rationalized the form of buildings based on contemporary notions of perspective, nature, and beauty. The framework was both systematic and generative and, more than any other work, is probably responsible for the rapid spread and evolution of Renaissance architecture [19].

The essential aspects of Alberti's work remained a dominant view of architecture until at least the mid-nineteenth century [19].

Although the rapid spread of Renaissance architecture was facilitated by the availability of a systematic and generative theory of design, it also benefited from a wonderful new technology for disseminating knowledge: the printing press. In fact, Alberti's *On the Art of Building* was the first printed book on architecture, preceding the printed edition of the more ancient treatise of Vitruvius by a year [25]. In light of this historical reflection, it is interesting to look at the field of intelligent tutoring systems. The theoretical language of the field is clearly framed by cognitive science and it is a framework that has the potential for producing systematic and generative accounts of the form of instruction. One of the most interesting aspects of the field is its view that technology--specifically the technology of artificial intelligence--can be used as a means for analyzing, disseminating, and communicating subject-matter knowledge. Can we make use of this theoretical framework and this view of knowledge dissemination by computational technology to place new views of instructional design into wide-spread practice?

In addressing this particular problem, I think it is important to keep in mind a significant constraint. For new practices to come into being, they often need to evolve from existing practice, or have the wide-spread and enthusiastic support of those grass-roots practitioners. Consider the history of educational technology. Time and time again, new technologies have shown great promise for revolutionizing instructional practice, and those promises remained unfulfilled. In the first third of this century, it was film and radio that would open new avenues to knowledge; in the second third, television; in the last third, computers. Cuban [4], in an analysis of this history, suggests that (a) practitioners naturally resist being forced to change, (b) they resist proposed changes they do not understand, (c) they resist changes that may affect their security, and (d) changes generated in a culture that values science and technology must be made understandable and valued for another culture. The upshot of this second historical lesson is that any new instructional design methodology must be somewhat consistent with current practice in the industry of instructional design.

Scope of This Paper

The main idea in this paper is that instructional design methodologies based on cognitive science perspectives can be developed along with supporting technological tools. One promising avenue is to begin to develop instructional design systems analogous to the computer-aided design (CAD) systems used in other design disciplines. The place where these are most likely to have an impact is in instructional design areas in which there is a high volume of instruction being continuously developed and in which the design problems have substantial overlap. Training in industry and the military are prime candidates for such developments. Instructional design in these areas very often involves the application of familiar analysis techniques and methods to new problems (e.g., maintenance training for a new device in a product line), the modification of existing instruction to meet changes in instructional situations (e.g., to meet changes in the student population), or the reanalysis of existing instruction to improve efficiency and effectiveness. Design problems in which there is a high carry-over of analysis techniques or design specifications from prior problems are well-suited for CAD-based methodologies.

A GENERAL FORMULATION OF DESIGN

Attempts to characterize the process of design has had a long history in cognitive science [21, 26]. Recently, Vinod Goel and I [8] used the following approach. Rather than attempt to come

to a specific definition of generic design, we described design as a radial category--that is, particular tasks will be called design to the extent that they match a prototype of the design process. Based on an information-processing account, we outlined how the invariants of the prototypical task environment for design interact with the invariants of human information processing to structure the problem space of design situations. This analysis was then related to data from verbal protocols from experts in instructional design, mechanical engineering, and architecture.

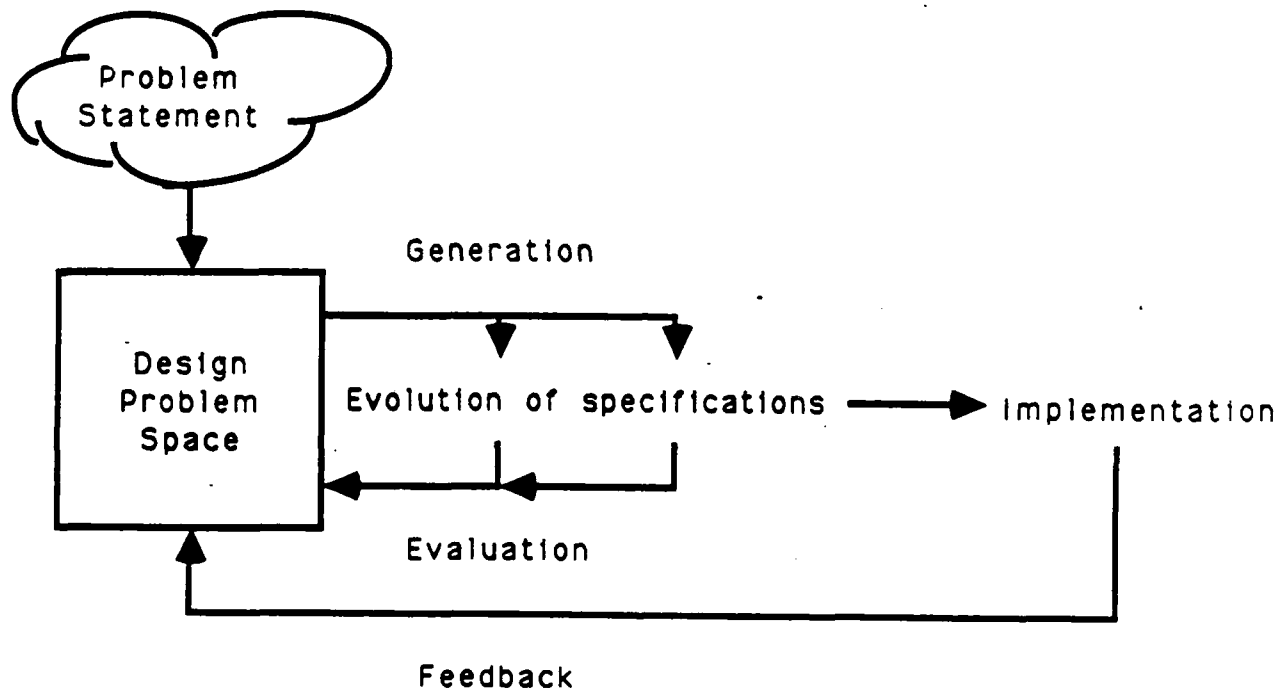


Figure 1. Structure of a prototypical design task.

Figure 1 provides a summary of the prototypical design task. A design problem is given, usually specifying the current state of the world and a set of intentions to be fulfilled by an artifact. In instructional design, the artifact may be a set of material resources, such as texts, along with some instructional process. The design process iteratively generates and evaluates a set of artifact specifications until they are deemed complete. Typically, these specifications go to another agent for interpretation and implementation. Also typical of this process, feedback from the world about the design of the artifact occurs after it has been implemented.

There are a number of important features of the typical design process that appear to have a substantial effect on design problem solving [8]:

1. There are many degrees of freedom or substantial lack of information in the problem definition.
2. There is delayed or limited feedback from the world during problem solving.

3. The input to design is largely comprised of goals and intentions. The output is the specification of an artifact.
4. The artifact must function independently of the designer.
5. The specification and implementation of the artifact are temporally quite separate.
6. Actions in the world have costs (e.g., time and money) and there are penalties for being wrong.
7. There are no absolutely right or wrong answers, but gradations of better and worse.
8. The problems tend to be large, complex, and require problem solving over the course of days, months, or even years.

Several entailments of these features were identified by Goel and Pirolli. Of particular relevance for discussion here are the following:

- Because there are many degrees of freedom in problem statements, design problems require substantial analysis, negotiation, and structuring.
- The delay of feedback, cost of action in the world, and the independent functioning of the artifact suggests that a substantial amount of performance modelling of the artifact must take place during problem-solving.
- The size and complexity of problems require management through problem decomposition and the correlated use of abstraction hierarchies in the specification of an artifact.

Knowledge structures problem solving and, in the case of design, theories about the form of artifacts and about the world influence the analysis and structuring of problems, the performance modelling of current specifications, and the kinds of abstractions and decompositions that occur during the design process. The term "theories" is used loosely here, for often it is some combination of engineering theory combined with personal rationalizations based on experience. Consequently, the ways in which design problems are solved are greatly determined by knowledge that shapes the designer's views of the world and the forms of artifacts that she intends to place in that world.

THEORIES OF INSTRUCTION

A fairly traditional view, is that instructional design theories attempt to specify the space of instructional *situations*, the space of instructional *methods*, and to develop statements, called *principles* or *theories*, that link these spaces. The analysis of instructional situations is taken to broadly include the effects of instructional methods, usually called instructional *outcomes*, and the *conditions* that affect the outcomes and use of those methods. Such conditions include the subject-matter, the instructional setting, properties of the targeted learners, and the nature of the learning task.

Traditionally, principles of instruction are taken to be those statements that characterize elementary building blocks for instructional methods. *Descriptive* principles are scientific

statements about the effects of a particular method under given conditions. *Prescriptive* principles are the kind of statement used in design to identify the optimal method to use in a given situation. Instructional theories deal with larger and more integrated sets of methods, describing outcomes and prescribing methods in given situations. Comprehensive instructional theories are intended to provide the knowledge base for solving the problems of instruction.

Elsewhere [18], we have looked at traditional views of the instructional world, such as work in the paradigm of Gagne and Briggs [7] and asked how that view is expanding and changing with developments in intelligent tutoring systems and cognitive science. The analysis of instructional situations has been enhanced greatly by richer and more detailed characterizations of the knowledge to be acquired by students and of the cognitive states of students throughout an instructional process. For example, Anderson's [2] review of kinds of expert modules in intelligent tutoring systems indicates the variety of ways of representing detailed knowledge targeted as instructional outcomes, including ways of representing declarative knowledge, procedural knowledge, and qualitative process models. Analysis of the conditions for instruction has been greatly enhanced by similarly rich and detailed student modelling techniques [30], which provide a means for tracking subtle changes in students' cognitive states during instructional interactions rather than broader characterizations of the student populations.

Commensurate with the evolution of more detailed ways of representing knowledge has been the evolution of techniques for the empirical analysis of behavior, largely deriving from work on verbal protocols [5, 12]. Research on interface design and tutoring strategies is expanding the space of instructional methods, and many authors [3, 29] have attempted to formulate principles to prescribe their conditions of use. Finally, there are attempts to produce tighter couplings of theories of instruction to theories of learning, and to develop systematic accounts of social patterns of activity and their relations to the development of meta-cognition, epistemology, and social practice.

These new ways of looking at instruction are the sorts of things that one would like to see structuring the problem solving of practicing instructional designers. One reason for this is that the new analytic techniques more accurately capture knowledge acquisition, performance, and transfer [27]. Another is that the new instructional methods can achieve at least a standard deviation improvement in efficiency and effectiveness over traditional group-oriented instruction. Unfortunately, there are no broad integrative theories of instruction arising out of the new cognitive science perspective. However, it may be possible to string together various bits and pieces of methodology into an effective whole. One way to do this might be to begin to develop CAD systems for instructional design practitioners.

COMPUTER-BASED INSTRUCTIONAL DESIGN SYSTEMS

Several computer-based systems for instructional design are under current development. Although none are in wide-spread use, it is useful to examine their form and function with an eye towards future possibilities. Two systems are discussed in some detail here (see also [31]). The first is ID Expert [17], which is the prototype for an expert system consultant for instructional developers. The second is the Instructional Design Environment (IDE), which is an augmented hypermedia system [23, 24].

ID Expert

ID Expert [17] is a prototype expert system intended for use as a consultation system by inexperienced instructional designers. It is currently implemented in an expert system shell, and contains about 400 rules. Use of the system assumes that the user has already done some analysis, involving the identification of student attributes, subject-matter knowledge, and the goals of the instruction. The user enters into a dialog with the system, eventually producing an output consisting of a set of specifications and recommendations for a course design.

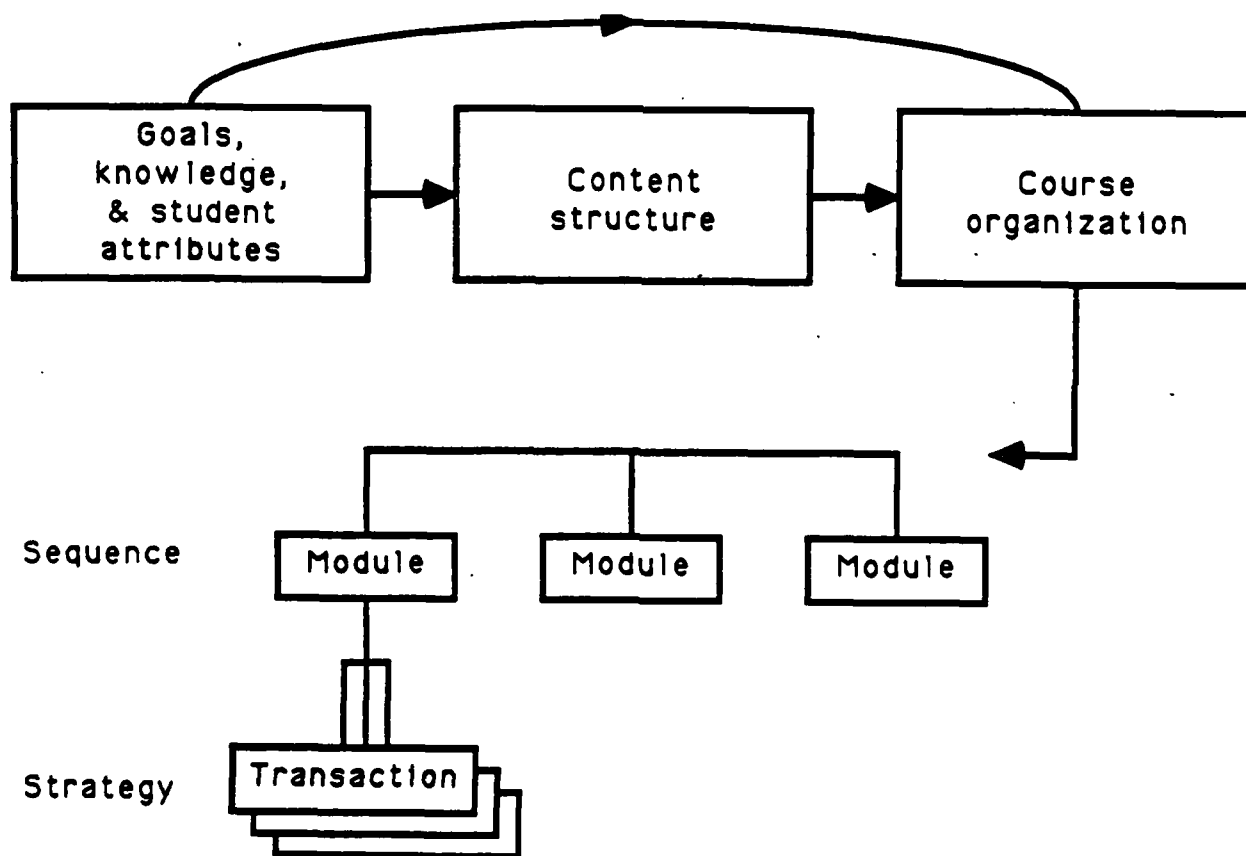


Figure 2. The model of instructional design underlying ID Expert.

It is apparent that the model of instruction underlying ID Expert derives from work in the Gagne-Briggs approach to instruction, and particularly on the more recent integrative theories of Merrill [16] concerning instructional presentations for specific concepts, principles, and skills, and of Reigeluth [20] concerning the organization of lesson modules and courses. Figure 2 illustrates the components of this model underlying ID Expert. The inputs to the process are specifications of instructional goals, subject-matter knowledge, and student attributes. Using these inputs, a *content structure* is constructed. A content structure is an organization of subject-matter content, and a particular organization will also depend on goals and student attributes. Based on the

content structure, goals, and student attributes, a *course organization* is constructed. This determines the paths a student may take through the components of instruction. This organization consists of *modules*, which are comprised of some *representation of content*, a set of *transactions*, and a set of *strategy rules*. Transactions are communicative actions or activities fulfilling some instructional function. *Sequencing rules* organize modules and strategy rules organize transactions.

ID Expert is in part dependent on a frame-based representation of various types of content structures, course organizations, strategies, and transactions. IDE Expert also has a rule-based component used to select and refine particular frames given that other frames have been selected and partially or fully instantiated. Although a substantial amount of knowledge has to be acquired from a user in order to construct a particular design, there is also a substantial amount of knowledge that is embedded within the system.

One interesting feature of ID Expert is that it not only knows what methods will achieve which goals, but it can also recommend a space of alternative methods with graded degrees of confidence across the alternatives. This is basically an extension of the technique of accumulating certainty factors in diagnostic expert systems. For example, to provide an overview, the system (on a scale of -1 to +1) might recommend a synthesis transaction with a certainty = .40, a summary transaction with certainty = .30, and an exposition with certainty = .10.

Although ID Expert is currently just a prototype for a consultation system it has some interesting properties:

- It suggests that substantial amounts of knowledge about instructional design can be embedded in an expert system.
- Such knowledge can be used to drive a designer through territory she is not familiar with, providing advice, suggestions, or even making decisions.
- The rationale for the design can be reconstructed by tracing through the rules suggesting methods to fulfill particular functions and the evidence accumulating for various alternatives.

The Instructional Design Environment

In contrast to ID Expert, in which the system drives the user, the Instructional Design Environment (IDE) centers on experienced users driving their way through the system. IDE [23, 24] is a hypermedia system in which instructional designers can enter, edit, and manipulate their analyses and specifications. Information is entered into *notecards* and relations among notecards can be specified by *links* of various *link types*. Notecards are of a variety of *card types*, and with each card type is an associated *substance*. Substances are essentially different kinds of media, such as text, graphical browsers, sketches, animations, etc., each coming with its own set of editing operations, and perhaps default content (for example, a set of slots to be filled, or icons that act as buttons to generate sets of actions). Notecards and links can be organized in *notefiles*, and notecards may contain *cross-links* across notefiles.

Figure 3 presents a conceptual view of a possible notecards structure in IDE, with notecards represented by boxes and ovals, and links represented by lines and arrows. At the top of Figure 3 is an analysis of subject-matter which organizes material around the functional decomposition of a

device into subsystems and parts, and associated analyses of relevant concepts and tasks. At the bottom of Figure 3 is part of a maze representing an abstract view of the structure of a piece of interactive video-disk instruction.

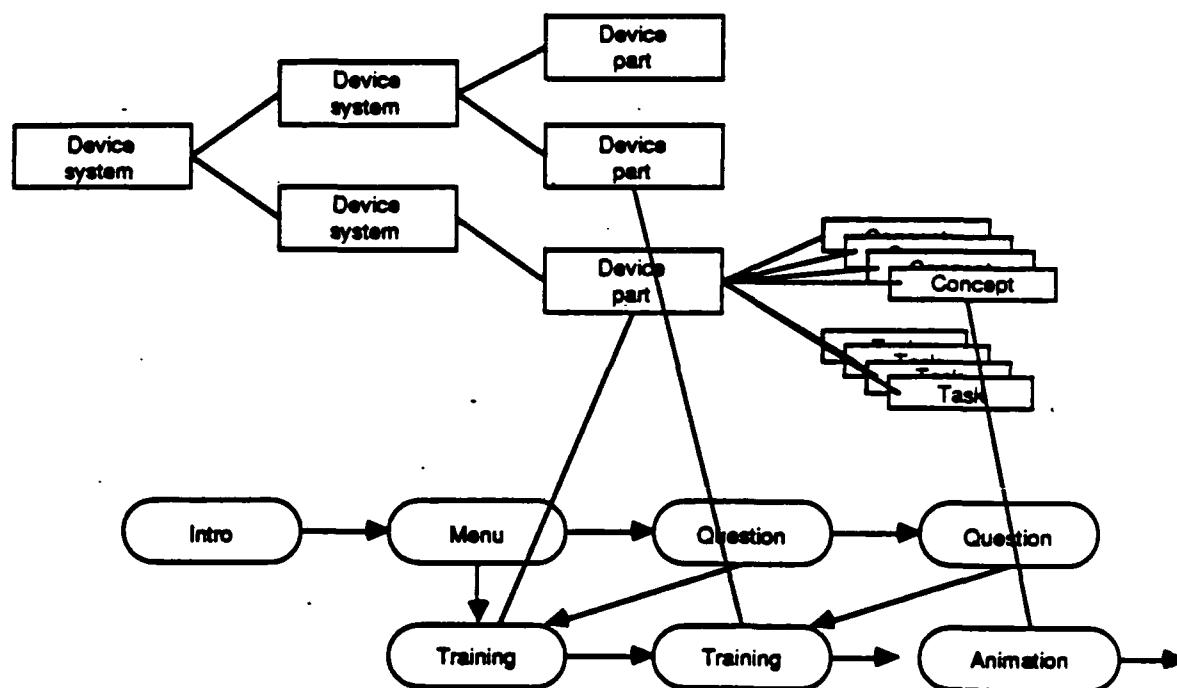


Figure 3. A heterogeneous mix of cards and links in IDE hypermedia analysis.

IDE is not coupled to any particular design methodology and consequently exists in a variety of forms, each tailored to the specific needs or interests of its users. Particular designers or groups of designers may tend to use specific ways of representing their analyses and specifications. To capture these regularities, new card-types are developed. In a fairly compelling way, these task-specific card types are analogous to the frame-based representations used in ID Expert. The card types are used to encode particular kinds of content representation (e.g., concepts, tasks, skills, device components) and various kinds of instructional presentations and organizations. Associated with particular representations are sets of common actions, which can be captured by menus and action-generating buttons associated with particular card types. These common operations are analogous to the knowledge encoded in the rule-based component of ID Expert. The construction of new card types and associated actions is available to designers through a relatively simple set of tools.

IDE also comes with capabilities for organizing related sets of card types and tools into *modes* that can be associated with the specific subtasks associated with design. For instance, in working with several instructional design groups [24], IDE developers structured IDE to have modes for: (a) data collection, (b) task analysis/structuring knowledge, (c) sequencing, (d) delivery, and (e) evaluation. The data collection mode is meant to be used during the early stages of design during which the main task is simply collecting incoming information and notes. The task analysis/knowledge structuring mode supports the development of analyses of subject-matter.

The sequencing mode is used to organize the content with respect to the instructional goals and student attributes. The delivery mode is used to sketch out, or actually specify the particular instruction in the media of choice. Evaluation mode is used to collect feedback from the implementation of the course.

IDE has been used in the design of instruction for a variety of media, including text, interactive video disk, and as a driver for an intelligent tutoring system. For example, in the case of interactive video disk, IDE was structured to contain card types and tools for specifying the commonly used abstractions for sketching out video disk sequencing (mazes, scripts), for prototyping frame sequences, and for actually driving the video disk [24]. Russell [22] describes the intelligent tutoring system architecture (IDE-Interpreter) that can take as input a set of specifications output from IDE. One could also view IDE-Interpreter as a facility for modelling the details of complex interactions that may occur in instructional interactions involving media that is not computer based.

The following are some of the interesting features of IDE noted by Russell et al. [24]:

- Although hypermedia is too ill-structured to be used as a true database management system, it can act as vast repository of heterogeneous information, and many kinds of queries can be answered through direct inspection or automatic search techniques tailored for specific tasks.
- Analyses, specifications, and other products are available in a sharable and inspectable form. In some cases, IDE collaborations have taken place across country.
- Permanent storage of structured and inspectable designs permits the reuse of analyses and specifications, the modification of materials to meet changing needs, and reanalysis of existing designs to extract principles or identify problems.
- The creation of representation types and tools implicitly creates a design standard and allows a design methodology to evolve through time.

Russell et al. also identify a host of research issues, which would actually have to be addressed in any hypermedia-based design environment:

- Kinds of analyses, design decisions, and rationale to record. As shown in recent research on software design [28], it is not always clear what sorts of information need to be recorded to facilitate future modifications or reanalysis, or use of components to solve new problems.
- Support of collaboration. Although IDE permits sharable notefiles, there are many issues surrounding versioning, means for keeping track of who is working on what, and negotiating the meaning of hypermedia structures among collaborators.
- Navigation and query techniques. As with all hypermedia systems, finding information and moving through complex structures often becomes a serious time sink.

ID Expert places much of the knowledge about the representation and process relevant to design inside the system, extracting the information needed to instantiate a particular design from a user who is an inexperienced designer. In contrast, IDE basically provides a fairly malleable

medium and it is up to the experienced designer-user to structure the process and fill in the relevant information. One could well imagine that having a set of expert system advisors or tools would be of use even to the expert, just as having an automatic symbolic integrator is useful to the engineer or scientist: to accomplish tasks that are well-structured but tedious, and to reduce cognitive load. It seems clear that an ideal instructional design system would sit somewhere between the system-driven approach of ID Expert and the user-driven approach of IDE.

LOOKING TO THE FUTURE

Although ID Expert, IDE, and other emerging instructional design systems show promise, none really meet our ideal of a system that incorporates substantial aspects of instructional theory from a cognitive science perspective. As noted in the discussion of IDE, the evolution of tools and representation techniques drives the evolution of new design standards and methodologies. What sort of tools and techniques are out there in the literature, ripe for the picking?

We can reconsider earlier discussion of the ways in which cognitive science has expanded and refined the traditional view of instruction. Refinements in knowledge representation techniques, and specifically their use in intelligent tutoring systems, is beginning to drive the development of a variety of general architectures for developing intelligent tutoring systems [14, 22]. The experience of some developers [2] suggests that such architectures will probably be somewhat specific to particular kinds of instructional problems. As with the IDE-Interpreter, such architectures could be coupled with design environments to facilitate the standardization of associated design methodologies and the development of databases of design knowledge. Even without complete tutoring system architectures, there has been substantial progress in formulating fairly general formalisms for representing knowledge in specific domains. For instance, Kieras and Polson [11] have developed a production system formalism for characterizing task-related knowledge and a generalized augmented transition network formalism for capturing device operations that can be useful in capturing man-machine interactions. Such kinds of detailed representation techniques would be useful even when the goal does not involve producing a sophisticated computer-based tutor.

Protocol analysis techniques have evolved to the point where they can be carried out in part automatically. The work of Means and Gott [15] illustrates a protocol analysis technique that can be used even when the analyst knows very little about the subject-matter area. The technique, called the two-expert method, basically involves having two experts develop problems and give them to each other. In the process, the expert gives verbal protocols of problem solutions, sketches relevant models, identifies relevant resources, and identifies alternative solution paths. Much of the work surrounds the development of a graphical representation of relevant portions of the problems space. Such techniques and associated graphical representations seem ideal for a computer-based medium.

The selection and organization of instructional content has received less attention than the analysis of expert and student knowledge, however there are a few interesting developments. Kieras [10] suggests a set of heuristics for developing a mental model to be taught in association with some task. Using a rational task analysis, domain experts, and relevant documentation, knowledge about how a device works is selected based on its relevance to task goals, the accessibility of system components to inspection or manipulation, and explanations of critical procedures. Lesgold [13] discusses an object-oriented approach to analyzing curriculum organization, in which constraints among content organization, student attributes, and instructional goals are resolved.

In sum, there are a variety of analytic and representational techniques under development that seem ready for incorporation in larger, more integrated approaches to the design of instruction. Traditional approaches to instructional design are admirable for their degree of completeness and coherence, however, to quote Alberti [1], "there is no reason why we should follow [the ancients'] design in our work, as though legally obliged; but rather, inspired by their example, we should strive to produce our own inventions, to rival, or, if possible, to surpass the glory of theirs."

REFERENCES

1. Alberti, L.B. (1988). *On the art of building in ten books*. (J. Rykwert, N. Leach, and R. Tavernor, Trans.). Cambridge: MIT Press. (Original work published c. 1450).
2. Anderson, J.R. (1988). The expert module. In M.C. Polson & J.J. Richardson (Eds.) *Foundations of Intelligent Tutoring Systems* (pp. 21-53). Hillsdale, NJ: Lawrence Erlbaum.
3. Anderson, J.R., Boyle, C.F., & Reiser, B.J. (1985). Intelligent tutoring systems. *Science*, 228, 456-462.
4. Cuban, L. (1986). *Teachers and machines*. New York: Teachers College Press.
5. Ericsson, K.A. & Simon, H.A. (1984). *Protocol analysis: Verbal reports as data*. Cambridge, MA: MIT Press.
6. Gadol, J. (1969). *Leon Battista Alberti, universal man of the early Renaissance*. Chicago University Press.
7. Gagne, R.M. & Briggs, L.J. (1979). *Principles of instructional design* (2nd ed.). New York: Holt, Rhinehart, & Winston.
8. Goel, V. & Pirolli, P. (in press). Motivating the notion of generic design within information processing theory: The design problem space. *AI Magazine*.
9. Grayson, C. (1979). Leon Battista Alberti architect. *Architectural Design*, 49, 7-17.
10. Kieras, D.E. (1988). What mental model should be taught: Choosing instructional content for complex engineered systems. In J. Psotka, L.D. Massey, & S.A. Mutter (Eds.) *Intelligent tutoring systems: Lessons learned* (pp. 85-111). Hillsdale, NJ: Lawrence Erlbaum.
11. Kieras, D.E. & Polson, P.G. (1985). An approach to the formal analysis of user complexity. *International Journal of Man-Machine Studies*, 22, 365-394.
12. Kuipers, B. & Kassirer, J.P. (1984). Causal reasoning in medicine: Analysis of a protocol. *Cognitive Science*, 8, 305-336.
13. Lesgold, A. (1988). Toward a theory of curriculum for use in designing intelligent instructional systems (pp. 114-137). In H. Mandl & A. Lesgold (Eds.) *Learning issues for intelligent tutoring systems*. New York: Springer-Verlag.

14. Mcmillan, S.A., Emme, D., & Berkowitz, M. (1988). Instructional planners: Lesson learned. In J. Psotka, L.D. Massey, & S.A. Mutter (Eds.) *Intelligent tutoring systems: Lessons learned* (pp. 229-256). Hillsdale, NJ: Lawrence Erlbaum.
15. Means, B. & Gott, S.P. (1988). Cognitive task analysis as a basis for tutor development: Articulating abstract knowledge representations. In J. Psotka, L.D. Massey, & S.A. Mutter (Eds.) *Intelligent tutoring systems: Lessons learned* (pp. 35-57). Hillsdale, NJ: Lawrence Erlbaum.
16. Merrill, M.D. (1983). Component display theory. In C.M. Reigeluth (Ed.) *Instructional-design theories and models* (pp. 279-333). Hillsdale, NJ: Lawrence Erlbaum.
17. Merrill, M.D. (1987). An expert system for instructional design. *IEEE Expert*, 2.
18. Pirolli, P. & Greeno, J.G. (1988). The problem space of instructional design. In J. Psotka, L.D. Massey, & S.A. Mutter (Eds.) *Intelligent tutoring systems: Lessons learned* (pp. 181-201). Hillsdale, NJ: Lawrence Erlbaum.
19. Raeburn, M. (1980). *Architecture of the western world*. New York: Rizzoli.
20. Reigeluth, C.M. & Stein, F.S. (1983). The elaboration theory of instruction. In C.M. Reigeluth (Ed.) *Instructional-design theories and models* (pp. 335-381). Hillsdale, NJ: Lawrence Erlbaum.
21. Reitman, W.R. (1964). Heuristic decision procedures, open constraints, and the structure of ill-defined problems. In M.W. Shelly & G.L. Bryan (Eds.) *Human judgements and optimality*. New York: Wiley & Sons.
22. Russell, D.M. (1988). IDE: The interpreter. In J. Psotka, L.D. Massey, & S.A. Mutter (Eds.) *Intelligent tutoring systems: Lessons learned* (pp. 323-349). Hillsdale, NJ: Lawrence Erlbaum.
23. Russell, D.M., Moran, T.P., & Jordan, D.S. (1988). The instructional design environment. In J. Psotka, L.D. Massey, & S.A. Mutter (Eds.) *Intelligent tutoring systems: Lessons learned* (pp. 203-228). Hillsdale, NJ: Lawrence Erlbaum.
24. Russell, D.M., Burton, R.R., Jordan, D.S., Jensen, A-M., Rogers, R.A., & Cohen, J. (1989). *Creating instruction with IDE: Tools for instructional designers* (Tech. Rep. PS-00076). Palo Alto, CA: Xerox Palo Alto Research Center.
25. Rykwert, J. (1979). Inheritance or tradition? *Architectural Design*, 49, 2-6.
26. Simon, H.A. (1973). The structure of ill-structured problems. *Artificial Intelligence*, 4, 181-204.
27. Singley, M.K. & Anderson, J.R. (in press). *Transfer of cognitive skill*.
28. Soloway, E., Pinto, J., Letovsky, S., Littman, D., & Lampert, R. (1988). Designing documentation to compensate for delocalized plans. *Communications of the ACM*, 31, 1259-1267.

29. Towne, D.M. & Munro, A. (1988). The intelligent maintenance training system.
30. VanLehn, K. (1988). Student modelling. In M.C. Polson & J.J. Richardson (Eds.) *Foundations of Intelligent Tutoring Systems* (pp. 55-78). Hillsdale, NJ: Lawrence Erlbaum. In J. Psotka, L.D. Massey, & S.A. Mutter (Eds.) *Intelligent tutoring systems: Lessons learned* (pp. 479-530). Hillsdale, NJ: Lawrence Erlbaum.
31. Wipond, K. & Jones, M. (1988). Curriculum and knowledge representation in a knowledge-based system for curriculum development. *Proceedings of the ITS-88*. Montreal, Canada.

LITERACY AS PREREQUISITE KNOWLEDGE

Glynda Ann Hull
Graduate School of Education
University of California, Berkeley
Berkeley, California 94720

SUMMARY

This paper argues that literacy skills--skills such as reading and writing which are used in interacting with print to function as a member of society--are prerequisite knowledge for users of intelligent tutoring systems. The author reviews recent assessments of the literacy skills of young adults; such assessments indicate deficiencies in literacy skills which require complex information processing. She also reviews recent research on the nature of reading and writing, in particular that which focusses on populations which are underprepared, and characterizes in more detail the kinds of literacy problems that students might bring to their interactions with intelligent tutoring systems. She illustrates some of these problems with reference to a tutor for teaching writing skills. She next discusses the changing nature of texts and the increasingly complex literacy requirements that will accompany new information technologies like hypertext and hypermedia systems.

INTRODUCTION

This paper presents some components from current literacy theory--ideas about the nature and function of texts and how people learn to read and write them--and asks how such ideas can inform the construction and use of intelligent tutoring systems. Although intelligent tutoring systems are sometimes created so that people can acquire complex knowledge and procedures in a context that is independent of reading and writing, I want to suggest that learning to use many such systems involves literacy-related activities, and that, for most learners, literacy demands will continue to increase as new contexts for use include new technologies, like hypertext and hypermedia systems. It is crucial, then, to pay careful attention to the literacy skills required to operate and learn from intelligent tutoring systems and other information technologies.

In offering these ideas, I attempt to represent the perspective of computer users who are students, to see through their eyes rather than through the lenses of developers or computer scientists or domain experts. In fact, I am particularly interested in those student users who are "underprepared" in literacy skills, whose reading and writing skills appear insufficient to accomplish the literacy tasks required by schooling or work. Research is showing that even those reading and writing performances which seem genuinely flawed possess a history and a logic. Such a perspective broadens considerably the population we think will be able to attain complex literacy skills.

In the first part of the paper, I describe the literacy problems that currently appear to be the most serious impediments for young adults in their schooling and the workplace. Then I review some recent research on literacy, work which characterizes reading and writing as complex cognitive and social processes, and

mention its implications for practice. In the next section, I discuss some of the literacy skills involved in using intelligent tutoring systems, with reference in particular to a tutor for teaching writing skills. Then follows a discussion of the changing nature of texts and the literacy requirements that will accompany new information technologies. I conclude with some recommendations for literacy instruction in our age of new technologies.

A LITERACY CRISIS

Everywhere there are warnings of a literacy crisis in America, warnings apparently buttressed by striking evidence. In the language of the National Commission on Excellence in Education [1],

- Some 23 million American adults are functionally illiterate by the simplest tests of everyday reading, writing, and comprehension.
- The College Board's Scholastic Aptitude Tests (SAT) demonstrate a virtually unbroken decline from 1963 to 1980. Average verbal scores fell over 50 points and average mathematics scores dropped nearly 40 points.
- Many 17-year-olds do not possess the "higher order" intellectual skills we should expect of them. Nearly 40 percent cannot draw inferences from written material; only one-fifth can write a persuasive essay....
- Business and military leaders complain that they are required to spend millions of dollars on costly remedial education and training programs in such basic skills as reading, writing, spelling, and computation (p. 26).

Although such warnings about insufficient language skills are frequently made, along with similar predictions about an illiteracy that is scientific or mathematical or historical or cultural in nature, many of these figures are regularly questioned, and their interpretation roundly debated. Indeed, the very language we have used to describe the problem--*literacy* as opposed to *illiteracy*--is now recognized as an inaccurate dichotomy. Most people are, in fact, illiterate in most areas of knowledge, but literate in a few, while literacy skills can more accurately be viewed to form a continuum. And though it is clear that some proportion of our U.S. population cannot read or write at all, and thus might accurately be termed "illiterate," that percentage is very low. Nor does the word *illiteracy*, with its connotation of an inability to sustain an independent and satisfying life, do justice to the ingenuity and attainments of many citizens who don't read and write well.

I make all of these caveates about our country's literacy "crisis," not because I believe there isn't one, or that we shouldn't be concerned about improving reading and writing skills, but because it is important to be clear about how we define and delimit the problem we most want to solve. I find some useful definitions and distinctions in a recent report released by the National Assessment of Educational Progress (NAEP, [2]). NAEP tested a national sample of 3,600 young adults on a range of test items representing typical written materials and information processing demands: reading bus schedules and tax tables, filling out application forms and check ledgers, deciphering road signs and grocery labels. According to this

assessment, only an estimated five percent of the population functions below the fourth grade level on literacy skills. However, around fifty percent of the nation's young adults is projected to possess what Thomas Sticht called "mid-level" literacy, skills which put them far above the literacy levels of preceding generations, but discouragingly below the skills needed in an information processing age. Kirsch and Jungeblut [2], the authors of the assessment report, concluded that:

It is clear from these data that "illiteracy" is not a major problem for this population. It is also clear, however, that "literacy" is a problem. Sizable numbers of individuals are estimated to perform within the middle range on each of the scales. Within these broad ranges, individuals are neither totally "illiterate" nor fully "literate" for a technologically advanced society (p. 5).

In their review of the NAEP assessment data, Venezky, Kaestle, and Sum [3] characterize the literacy crisis similarly. They see a subtle danger in the failure of young adults to be competent problem solvers. For example, they point out that respondents in the NAEP study did well so long as they were asked to identify a single item of information from a prose passage and the language used to describe the item in the question and the text were the same or almost the same. However, when the information couldn't be derived from a single sentence, but rather had to be drawn or inferred from several sentences or a passage, respondents did less well. Similarly, respondents could use an index accurately when that process required only one step, but were derailed by multi-step searches. Difficulty with items involving computation also seemed to center on the complexity of print processing rather than arithmetic operations alone. These NAEP data suggest that most young adults have decent abilities when it comes to basic tasks, but fail to thrive when tasks require more complex problem-solving. And among those who do poorly, a disproportionate number are racial and ethnic minorities--Blacks and Hispanics. If current demographic projections hold, the proportion of young adults who have insufficient literacy skills will increase in future years.

There is a literacy problem in our country, then, but this problem isn't for most young adults so much a crisis of "basic" skills--being able to decode individual words and spell them correctly. It has to do, rather, with literacy put to more complex purposes, as would seem fitting for a society in an information age. There is some debate, however, about whether our technological society will continue to require increasing numbers of future workers who possess high-level literacy skills. Some argue that white collar jobs will be downgraded, and others, that white-collar jobs which require problem-solving and adaptable learning are bound to increase. (See, for example, Sticht [4]). In a later section, I will suggest that new information technologies have the potential to put more information into the hands of more people, and that the mark of a literate person will increasingly be measured by the capability to manage that information in more sophisticated ways than we currently are able. However, before discussing new conceptions of literacy suggested by these technological advances, I'll review current theory on writing and reading.

WRITING AND READING AS COGNITIVE AND SOCIAL PROCESSES

There has been much research in recent years on reading and writing as cognitive acts. The work on reading characterizes this process as "constructive." While it was formerly believed that reading consisted of recognizing one word, and then another and another, and finally combining those words into meaningful sentences and larger groupings, current theorists emphasize the degree to which what a reader understands from a text greatly depends on the knowledge he or she brings to the text. That knowledge includes general information about the world, assumptions about the author's intentions, knowledge about genre and particular discourse conventions. We once thought of texts merely as repositories of information where any good reader would withdraw the same set of facts and knowledge. The contrasting current view is that "a text is not so much a vessel containing meaning as it is a source of partial information that enables a reader to use already-possessed knowledge to determine the intended meaning" ([5] *Becoming a Nation of Readers*, p. 8). Better readers connect the information in texts with the knowledge they already have, while poorer readers may overemphasize the text or their own knowledge. Unlike poorer readers, better readers are strategic, varying how they read according to how difficult the text is, their purpose for reading, and how much they know about the topic. And they monitor their own comprehension, taking corrective action once they recognize that a text is not making sense.

Research on writing from a cognitive perspective views writing as a "problem-solving" process, a process whereby writers engage in conscious cognitive and linguistic behaviors [6]. Researchers have found that writing consists of several main processes--planning, transcribing text, and reviewing--and that these processes don't occur in a particular order. Instead, one process can interrupt another: a writer can stop to plan in the middle of transcribing a paragraph; a writer can begin to revise before he has a word on the page. Better writers develop guiding plans for producing texts that are flexible--amenable to revision. On the other hand, poorer writers don't plan as much as they should and then don't want to discard plans they have constructed even when those plans are dysfunctional. Better writers spend more time improving the meaning of their texts, while novices are more likely to limit themselves to changes that improve surface features such as spelling or word choice.

Most recently, theorists have begun to view reading and writing not only as complex cognitive processes, but as processes embedded in particular contexts. This view suggests that whatever skills we value as desirable literacy performances will depend for their meaning and practice upon social institutions and conditions. It makes sense, then, to think about writing not as a single process, but as a plurality. What counts as good writing will vary, depending on the function that writing will serve and the audience it is for. We are also beginning to think about learning to read and write as enculturation into a community or a discipline. Being a good reader in a literature class means, then, more than being able to decode sentences and to summarize the content of an essay. It includes, in addition, learning the values and ways of knowing that characterize the discourse of a particular community [7].

This research suggests some implications for practice. People most easily acquire literacy in settings which provide authentic tasks and the scaffolding necessary to accomplish reading and writing tasks that are too difficult to handle

G. Hull

4

alone. One of the most important factors in literacy instruction for underprepared students is a recognition that learners' performances, though they may seem very flawed or idiosyncratic, possess a history and a logic. Learners are capable of much more than our interpretation of their reading and writing performances lead us to think. However, they are likely to bring misconceptions to literacy tasks

LITERACY AND INTELLIGENT TUTORING SYSTEMS

It's popular today to add new literacies to the list that educated citizens are supposed to acquire, such as "computer" literacy or knowing how to operate various systems and to understand the functions that those systems can serve. I want to discuss here, not the new computer skills that people must acquire if they are to become literate or proficient at using intelligent tutoring systems, but the literacy skills and literacy-related activities that users may engage in when learning to use and when operating intelligent tutoring systems. I will refer particularly to a tutor for teaching learners to correct errors in their writing.

There have been few efforts to build intelligent systems for teaching reading and writing skills because of the formidable problem of natural language understanding. A system which could process unrestricted input--which could, for example, read any document and assess its propositional accuracy--is a wonderful but unattained (and some would argue unattainable) vision. However, there have been several attempts to build systems which analyze written language in ways that can be instructionally useful to writers (e.g., WRITER's WORKBENCH [8], CRITIQUE [9]).

To build one such system, MINA (Misconceptions are NATural), I and other researchers at the University of Pittsburgh created a taxonomy of the common errors that young adult writers made in their writing and then constructed a program that would identify some of those errors by flagging designated error patterns [10]. The system also contained a tutor which led users to consider their error patterns and how they might correct them. Specifically, writers saw particular error patterns highlighted in their own writing on the screen, and they were encouraged to use on-line helps to identify and correct them. These helps included a set of written examples and definitions and correction tips and also a speech synthesizer whereby users could hear sentences read aloud. The system would also delimit the amount of text a user had to scan in order to detect an error--first highlighting a few lines of text as potentially errorful and then narrowing the highlighting to the actual errors.

Testing MINA with students revealed some interesting things about the process of learning to correct errors. It showed us, for example, the fragility of knowledge structures: students rarely acquired new rules about sentence construction cleanly and robustly but learned through slow stages of successive approximation. We also came to recognize that particular literacy skills were required to operate and learn from the program, skills apart from the ones the program purported to teach. As mentioned above, one of the help levels that our system offered was a set of definitions and examples and tips. We took great care in constructing this information--avoiding grammatical terminology, using simple language in our examples, linking one concept to another, predicting pitfalls common among beginning writers. We found, however, that students used this help level infrequently and poorly. One reason for this no doubt had to do with a general tendency on the

part of computer users to disregard on-line help. Another reason, however, centered on poor reading skills. We noticed that when students took the time to read our help files, they misread or misinterpreted much information. Another reason had to do with users' simply not knowing how to use the help files--where to look for examples or when to expect and seek more information.

As a result of these testing sessions, we came to think of our tutor differently. It was not the case that we had created a self-contained and autonomous environment in which users needed no skills to learn except those we had already provided the scaffolding for. On the contrary, students brought with them to our tutor poor learning skills acquired elsewhere, and these skills of course helped to determine whether they would be successful with our tutor. Students were also faced with a new learning environment--one that required them to be facile at locating relevant information and to be metacognitively aware of when new information was required. We came to think of our the activities surrounding the use of our tutor as comprising a special kind of literacy event, the conventions for which could never be transparent but which, rather, students would need to learn. In other words, in constructing our tutor and in introducing it to users, we were more aware of the cognitive demands of the tasks than the social context in which those demands are played out. I want to argue for the importance of both in introducing users to intelligent tutoring systems.

THE CHANGING NATURE OF TEXTS: HYPERTEXTS, HYPERMEDIA, AND NEW LITERACIES

So far I have limited my comments on literacy and computer tools to traditional current views of reading and writing. However, with the advent of new information technologies, the nature of texts is changing, and with those changes will come new demands for complex literacy skills. For many years, when we have thought of texts, we have had a particular kind of text in mind: a text written or printed on paper, read from beginning to end, and comprised mostly of words with perhaps illustrations or graphics as augmentation. In the world of business and industry and science, we have also expected expository texts to display particular kinds of logic and reasoning. A characteristic of all books is that they are static: once it is printed, a book doesn't change, and readers certainly don't have the opportunity to manipulate its contents [11]. But with new information technologies--usually called hypertext and hypermedia systems--our notions of what a text can be are changing, along with our understanding of what it means to read and write.

Hypertexts are electronic documents that can be non-linear. That is, instead of being read from beginning to end, as is customary with printed matter, hypertexts allow a reader to chart his/her own path through information. One screen of a hypertext document might be a conceptual map of the topics contained within the document. One reader might, then, peruse this map and choose one starting place, given his own interests and purposes in reading; another reader would most likely explore the information in a different order. A hypertext can be non-linear in other ways as well. Suppose, for example, that a reader comes upon a concept in the course of reading that she needs further information about or a topic that she wants to pursue. In an appropriately constructed hypertext, that reader could indicate her decision, whereupon the screen would reopen or a new window would appear

presenting the desired information. Imagine a film in the window or sound or an animation or another graphic, and you have an example of hypermedia.

When technology enthusiasts talk about the advantages of hypertext and hypermedia systems, they often mention "connectivity" [11]. Such systems are well suited for allowing writers to indicate connections among ideas and readers to explore those connections. They also praise the interactive nature of documents created with such systems. It is possible, for example, for readers and authors to add to and alter electronic texts in ways impossible with paper ones. Hypertexts can be customized for different audiences. A hypertext of a textbook or training manual could contain several versions of the same information, and a reader could select the level of reading difficulty appropriate for him. People also think that hypermedia offers the advantages associated with different modes of presenting information. Certainly complex procedures can often be described through film or animation rather than or in addition to text, for example.

Developers are beginning to create data bases in the form of hypertext and hypermedia systems. Such data bases are particularly interesting to educators, who see the potential to transform schooling by virtue of bringing information resources to the classroom where that information can be manipulated and explored in ways that haven't normally been possible previously. (See, for example, the hypermedia projects on Shakespeare [12] and Steinbeck's *Grapes of Wrath* [13]). Indeed, at the center of the literacy activities that will characterize the exploration of data bases is information management: deciding what information is important, how to synthesize it, how to organize it, how to interpret it, how to represent it. The dream of hypertext enthusiasts is not just self-contained data bases, however, but electronic access to all sources of information in museums and libraries and data bases and publications of all kinds. The skills required to manage information in such a world will be at a premium.

Research on the skills required to construct and process hypertexts is just beginning [14]. But it seems certain that these skills will differ from traditional literacy skills and will exceed them in complexity. The skills required go far beyond merely comprehending a single text or producing a coherent text for a single audience; they also involve being able to select relevant information from abundant resources and connect that information to other knowledge, possibly through multiple modalities. It is discouraging, then, when we recall the recent NAEP data on the literacy skills of young adults which suggests that the skills these respondents were weak in were those that required more complex information processing, that required, for example, multi-step procedures. On the other hand, we can be cheered by the recent understandings of literacy, which give us some hints about the transmission of reading and writing as cultural and cognitive skills, and recent research on underprepared students, which suggests that their abilities exceed our expectations.

LIST OF REFERENCES

1. National Commission on Excellence in Education. (1983). *A nation at risk: The imperative for educational reform*. Washington D.C.: US Department of Education.

2. Kirsch, I., & Jungeblut, A. (1986). *Literacy: Profiles of America's young adults*. Princeton, NJ: National Assessment of Educational Progress.
3. Venezky, R.L., Kaestle, C.F., & Sum, A.M. (1987). *The subtle danger: Reflections on the literacy abilities of America's young adults*. Princeton, NJ: Center for the Assessment of Educational Progress.
4. Sticht, T. G. (1987). Literacy, cognitive robotics and general technology training for marginally literate adults. In D. Wagner (Ed.), *The future of literacy in a changing world* (pp. 289-301). New York: Pergamon Press.
5. Anderson, R.C., Hiebert, Elfrieda H., Scott, J.A., & Wilkinson, I.A.G. (1985). *Becoming a nation of readers: The report of the Commission on Reading*. Washington, D.C.: The National Institute of Education.
6. Flower, L., & Hayes, J.R. (1980). the dynamics of composing: Making plans and juggling constraints. In L.W. Gregg & E.R. Steinberg (Eds.), *Cognitive processes in writing* (pp. 31-50). Hillsdale, NJ: Lawrence Erlbaum.
7. Hull, G.A. (in press). Research on writing: Building a cognitive and social understanding of composing. In L. Resnick & L. Klopfer (Eds.), *Cognitive research in subject-matter learning*. 1989 Association for Supervision and Curriculum Development Yearbook.
8. Frase, L.T. (1983). The UNIX Writer's Workbench software: Philosophy. *The Bell System Technical Journal*, 62, 1883-90.
9. Richardson, S.D. (1985). Enhanced text critiquing using a natural language parser. Yorktown Heights, NY: IBM Thomas J. Watson Research Center.
10. Hull, G., Ball, C., Fox, J.L., Levin, L., & McCutchen, D. (1987). Computer detection of errors in natural language texts: Some research on pattern-matching. *Computers and the Humanities*, 21, 103-118.
11. Yankelovich, N., & Meyrowitz, N. (1985). Reading and writing the electronic book. *Computer*, 18, 15-29.
12. Friedlander, L. (1988). The Shakespeare project. In S. Ambron & K. Hooper (Eds.), *Interactive multimedia: Visions of multimedia for developers, educators, and information providers* (pp. 115-41). Redmond, WA: Microsoft Press.
13. Campbell, R., & Hanlon, P. (1988). Grapevine. In S. Ambron & K. Hooper (Eds.), *Interactive multimedia: Visions of multimedia for developers, educators, and information providers* (pp. 157-177). Redmond, WA: Microsoft Press.
14. Charney, D. (1987). Comprehending non-linear text: The role of discourse cues and reading strategies. *Hypertext '87 Papers*.

Technology Assessment: Policy and Methodological Issues

Eva L. Baker

UCLA Center for Technology Assessment and Advance Design Information

ABSTRACT

How can we promote technology to its most effective use in training environments? The paper will explore the role of technology assessment in answering that question. Technology assessment will be defined, and its functions and future will be contrasted with typical perceptions of evaluation. Technology assessment divides into two major areas: (a) assessment of particular cases of the technology, and (b) assessment of the collective set of applications and predictions about future utility. Technical and practical problems in conducting either type of TA will be discussed. Major issues involve the outcome measurement, use of quality indicator models, the mixed models of assessment, the essential social character of the assessment, and problems of reporting. Short-term and long-term research and development implications will be suggested.

INTRODUCTION

How do we assess the utility of technology in training? How should information be assembled, analyzed, and interpreted to promote defensible decisions on the use of technology in training environments? In addressing these questions, this paper focuses on the interplay between methodological and policy issues. For at least five reasons, the term technology assessment is deliberately used instead of evaluation, a more common descriptor of the process described above (see, for example, Alkin, Daillak & White, 1979). First, evaluation, as it employed in education and training environments, connotes to many a relatively narrow set of methodological choices. For instance, it is common to assume that evaluations necessarily have certain features. Evaluations appear, for example, to be empirical in nature and obligated to: (a) collect data using designs similar to those employed in experiments, i.e., control groups, (b) use quantitative analysis as the basis for inference, and (c) focus on summarizing and reducing data. Viewing evaluation as bound by a relatively constrained methodology is an inaccurate but relatively widespread perception. A second, venerable misunderstanding of evaluation is the belief that its purposes are either summative or formative. Believers in this analysis assign studies into either decision or program improvement slots, as if such functions were mutually exclusive. A third, and seriously limited conception of evaluation can be traced to the systems approach underlying most evaluation models. Such models almost always focus exclusively

Eva L. Baker

1

on the performance of the intervention against relatively simple requirements (i.e., desired performance objectives). Fourth, evaluation studies normally apply to individual cases, such as one tutor, rather than a class of technological uses and functions (e.g., intelligent tutoring systems). When focused on one particular specific implementation, evaluation purposes may also become suspect. Evaluation is inferred to be a politically inspired process.

CHARACTERISTICS OF TECHNOLOGY ASSESSMENT

Technology assessment should supplant evaluation if only to avoid the enumerated liabilities of an older term. But the concept of technology assessment is appropriate to our present discussion for a number of important, additional reasons. Consider first that training technology itself differs fundamentally from other instructional interventions. Technology is interactive, dynamic, and develops rapidly, often in surprising leaps and directions. Paradoxically, the power of technology continues to expand as its costs, with relatively small bubbles, continue to drop. Thus, to think of technology as simply another delivery system, comparable to lecture-discussion, is to miss the conceptual boat.

This analysis implies that decision-makers should not focus alone on short-term yacht races between one instructional delivery system and another. When new technology first gets built and evaluated, it usually fares poorly in comparison to well-established practical alternatives, such as lectures and books. Thus, initial effects are almost always underestimated. Rather, studies of technology must be especially sensitive to the notion of technology-push (Glennan, 1967), the idea that technology bumps up against the usual requirements-driven programs in odd and unexpected ways. For what technology is almost guaranteed to do is to generate, by its very existence, outcomes and applications that were not previously considered by the training system, nor imagined by the technology designer. These new uses may be mistakenly described as side effects, when, in fact, they may be the delayed but central outcomes of the innovation. A critical element in technology assessment, therefore, is identifying when these options represent powerful, useful approaches, goals, or recombination or redefinitions of prior goals. As a corollary, new technology, more than other sorts of innovation, should not be shut off because its superiority on existing goals cannot be immediately demonstrated. For example, one effect of designing tutors may be the development of technology to create new kinds of human performance measures (Baker & Linn, 1985; Lesgold, Bonar & Ivill, 1988; Collins, 1987) and new ways of conceiving of performance tasks (Means & Gott, 1988). It is possible that such practical and conceptual outcomes may be more important than the adaptive wonders of instruction that particular intelligent systems are purported to create. If we are to develop clear traces of the broad utility of technology to meet training needs, studies must involve analyses far beyond what the technology designer or any given set of trainees believes or

Eva L. Baker

experiences. Policymakers need to be involved actively and early to determine which options should be highlighted, tracked, and ultimately ratified as bonafide new goals and functions.

The detection of the unforeseen has fundamental costs that policymakers should consider. These costs involve changing expectations about the purpose of studies. Minimally implied are a period of suspended disbelief and a planned commitment to the conduct and analysis of a network of studies of individual cases of technology. Because it takes time to execute such studies, they cannot be the sole initiative of an individual with only a limited period of assignment. Some larger, longer-term policy must be put in place. To reiterate, the purposes of such investigations focus on not only the differential impact of particular instances—tutor A versus option B—for particular tasks, but the larger and more important task of forecasting the utility of a class of technology. Thus, the explicit goals of technology assessment are dual: the case, usually against a specific training requirement; and the class, forecast for known and uncertain future requirements.

There is an additional, methodological nuance of the term assessment. It also implies a broader conception than typical product evaluations, specifically attention to multiple measures of input, context, and conditions of implementation, as well as to specific requirement-driven and unforeseen outcomes. The goal of such activity is to develop training quality indicators that provide composite estimates of variables and the relationships among them, much as economic indicators provide composite descriptions/forecasts. This indicator assessment perspective, by the way, is apparently useful to state policymakers, legislators, governors, and educational boards and superintendents, as they try to determine systematically and longitudinally the systemic consequences of policy changes to improve the quality of precollegiate educational services. (See, for instance, U.S. Department of Education Office of Educational Research and Improvement, 1988). Note that multiple indicator development is a natural opportunity for collaboration among branches and services that assess technology. Yet, full-blown technology quality indicator systems are still a long way off. Many specific problems must first be addressed.

TECHNICAL AND PRACTICAL PROBLEMS

This section of the paper addresses a litany of some of the major practical and technical issues that must be confronted to improve the information base for decision-making in technology and training.

Outcome Measurement

Even the most limited assessment needs high quality outcome data to contribute to judgments of present and predicted impact. Relatively few studies of intelligent systems use outcome data of any sort (Anderson, in press, is an exception). Other designers describe system goals for learners and then, at best, measure a proxy of performance. In two studies of tutors at UCLA (Baker, Aschbacher, Feifer, Bradley & Herman, 1985), we attempted to design, using domain-referenced procedures, measures of only those goals articulated by the designers. For example, in a study of WEST, we developed measures of computational skill, referenced to the universe of problems that students confronted, as well as measures of game strategy, an articulated but surprisingly unmeasured outcome. It is our view that outcome measurement of complex training goals is so important that it cannot be left to the designer alone to accomplish. Designers do not always work with teams who have measurement experience, so expertise may be missing. Secondly, important training outcomes need multiple measurement across time and conditions to consider effects beyond the short-term achievement of the training goal. These dimensions include retention, robustness of performance across field conditions, transfer, and assessment of underlying constructs to facilitate cross training, to name of few.

New developments in performance assessment lean heavily on trainee-generated performance, or constructed responses, using constructs from cognitive learning theory to derive scoring attributes (Glaser, in press; Baker, in press). The major message of this development is that measures must map back to characteristics of learning, i.e., elaboration, schema, and problem detection. Such approaches are partially validated by using expert-novice distinctions. This concern with the close relationship between learning and measurement conditions contrasts strongly with the majority of current outcome testing practice, where convenience and simplicity of test formats strongly influence what we are able to say about student performance. A second direction in performance assessment (Baker & O'Neil, 1989) involves improved, more sensitive ways to select and train judges of performance. No longer is simple designation as a subject matter expert, for instance, sufficient to assure reliable and valid assessment.

Mixed Models of Assessment

Remember that the term evaluation still calls up for many the specter of a single, monolithic methodology, largely social science derived, experimental, and quantitative in nature. That view may have accurately characterized the majority of social science research twenty years ago, but only accounts for a limited proportion of current effort. It is true that the field of evaluation a dozen years ago

Eva L. Baker

was fractionated into methodological camps, with lines drawn between quantitative experimentalists and qualitative interpreters. However, at present, we have *glasnost*, a more balanced blend of eclectic methodology. For example, it is common to mix relatively objective forms of performance assessment with qualitative analyses of protocols of trainee thinking (Feifer, 1989). Intensive descriptions of processes—for instance, knowledge engineering (Baker, Novak & Slawson, 1989)—and understanding queries in natural language systems (Baker & Lindheim, 1988), can be combined with surveys, and more traditional test forms to provide a more complete explanation for findings.

Thus, dichotomies such as cognitive versus behavioral, qualitative versus quantitative, descriptive versus experimental, and formative versus summative no longer present real choices. Rather, methods are selected to provide insights as appropriate and to provide multiple measurement of the same construct—a fact that is desirable both for the development of indicators, and for the conduct of serious validation studies.

Social Character of Assessment

When enumerating misunderstood issues in evaluation at the outset of this paper, the inferred political impetus of evaluation was briefly described. The evaluatee believes evaluation is an instrument of aggression (for evaluation occurs when someone has a problem). Program managers, on the other hand, may use evaluation defensively. They primarily may be interested in it as a defense against future assaults, rather than in the information it provides about innovation. Most efforts at assessment or evaluation provoke some level of resistance, resentment, or defensiveness. No one really believes the slogan "we're here to help," and they are often correct. Over and above usual paranoia, there are issues in the social context of new technology assessment that deserve comment, since much assessment is a social as well as a technical enterprise. One issue is: Who does the assessment for what ostensible purpose? If the designer is responsible for assessment, one is not only limited to a particular vision, but also inevitably confronted with self-interest. Furthermore, designers are more committed to the task of creating systems than to creating systems that result in demonstrable trainee outcomes. Particularly in computer-based technologies, the trick is to make a system run according to prediction. The importance of process is highlighted in an article by Cohen and Howe (1988). As they describe evaluation, it is limited to an expert review of the quality or process of research efforts. This article was especially heartening for me because it confirmed an earlier conclusion about the distinctions among expectations of high technology researchers, program managers, and evaluation and assessment professionals, and the resulting social complexity of getting the job done.

In emerging fields, researchers may propose to create a training system, program managers may think that's what they bought, and those charged with assessment may assume that training outcomes should be measured. In fact, the likelihood of strong outcome measurement increases only with the maturity of the technology. With nascent technology, the designer's claims and focus on a training system may simply provide constraints on the selection of a research problem. The researcher may say, and believe, that the chosen task is to develop an intelligent tutor to teach specific outcomes. But what the researcher may mean is that research will be conducted on an interesting part of the problem of developing a tutor. Researchers are not the same as training system developers, and this fact is demonstrated recurrently by the number of partial systems: tutors without student models, tutors, with wonderful diagnostic capability, tutors without pedagogical modules. Awareness and understanding of the various contexts and subtexts of communication among researchers, managers, and assessors may allow some form of collaborative assessment to work. It is also undoubtedly useful to program managers to understand underlying messages, particularly when they may have obtained priority for funding a particular technology program with a promise of a product for an actual training system. Tolerance for exploratory behavior seems to be directly related to size of budget.

A second, more obvious difficulty, especially within newly emerging fields, is the insider/outsider problem. We are experiencing this phenomenon in our DARPA project on assessing AI systems. Expertise and expectations differ, suspicions abound, not only between measurement specialists and AI researchers, but among linguists, psychologists, and AIs, and within the AI community between devotees of one or another approach. We have tried numbers of options to bridge the communication and knowledge gaps, including hiring AI people, providing incentives, using consecutive translation, and throwing ourselves on the mercies of friends. The trade-off here again is objectivity and detachment versus credibility and insight. A solution, of course, is to train people who become proponents and experts in the assessment of technology, but that will only happen when the technology has a surplus of researchers—a self-contradictory state when the focus is new technology. Yet, collaborative teams at UCLA, at LRDC, and elsewhere are at work. How they forge successes should be an interesting story.

Reporting

A final, and overlooked, area is the nature of reporting useful information for various levels of program and policy decisions. The identification of the full range of audiences is a critical point, as is the understanding that any data or conclusions can be used or misused against you. The challenge is to find ways of communication that will contextualize results appropriately, without endless qualification, reams of tables, or micromud descriptions that put off all but the most devoted reader. One

Eva L. Baker

area of general interest is trying to get a handle on the report users' mental models and effective decision options. If we could apply what we know from cognitive psychology to assist sophisticated decision-makers process and integrate assessment findings, we would develop clues related to what information was most relevant. Furthermore, one might expect that such report readers might themselves need a modicum of training so that we could assure that more than one reader would reach the same set of conclusions given similar findings.

RESEARCH AND DEVELOPMENT IMPLICATIONS

Short Term R & D

The issue enumeration above leads directly to some recommendations for R&D activities to advance the field. First, in the general area of technology assessment of intelligent tutoring systems, it will be important to categorize systematically the existing and developing Defense supported tutors by attributes, technical approach, and training tasks. UCLA has undertaken this task for DARPA in the area of natural language (NL) understanding systems and has created a sourcebook of the problem types that natural language systems address (Read, Dyer, Baker & Butler, 1989). Reviews of the utility of this sourcebook by researchers in the field have been very positive. The NL sourcebook is also available for use in a database form which can be updated.

A second short-term project involves the creation of advisory or assessment authoring systems particularly suited to technology problems. A prototype system has been developed at UCLA on the narrow problem of reliability for criterion referenced tests, and costs for a library of such aids are relatively small.

A third activity might be a case study analysis of an attempt at class-oriented technology assessment of ITSs, using naturally occurring and planned assessments.

Fourth, decision-maker mental model research could be conducted to provide better understanding of assessment requirements.

Long-term Studies

The design of seriously planned embedded assessment systems that includes the full range of input, process, and outcome data, such as individual differences, process, trainee performance, retention, and transfer data could be undertaken in a long-term study.

SUMMARY

This paper introduced the topic of technology assessment and tried to illustrate the benefits of a broader policy-sensitive approach. Specific problems in data quality models and social context of assessment were discussed. Finally, R&D options in this area were provided.

REFERENCES

- Alkin, M.C., Daillak, R., & White, P. (1979). *Using evaluations: Does evaluation make a difference?* Beverly Hill, CA: Sage Publications.
- Anderson, J.R. (in press). Analysis of student performance with the LISP tutor. In N. Fredericksen, R. Glaser, A. Lesgold, & M. Shafto (Eds.), *Diagnostic monitoring of skill and knowledge acquisition*. Hillsdale, NJ: Lawrence Erlbaum Associates.
- Baker, E. L. (in press). Cognitive approaches to measure deep understanding of subject matter. In M.C. Wittrock & E.L. Baker (Eds.), *Testing and cognition*. Englewood Cliffs, NJ: Prentice Hall.
- Baker, E.L., & Lindheim, E.L. (1988). *A contrast between computer and human language understanding*. Los Angeles: UCLA Center for the Study of Evaluation.
- Baker, E.L., & Linn, R. (1985). *Institutional assessing and improving educational quality* (Proposal to the National Institute of Education for the Center on Student Testing, Evaluation and Standards). Los Angeles: UCLA Center for the Study of Evaluation.
- Baker, E.L., & O'Neil, H.F., Jr. (1989). *Taxonomy of performance assessment*. Sherman Oaks, CA: Advanced Design Information, Inc.
- Baker, E.L., Novak, J., & Slawson, D. (1989). Feasibility study of an AI testing advisor (Report to OERI). Los Angeles: UCLA Center for the Study of Evaluation.
- Baker, E.L., Aschbacher, P., Feifer, R.G., Bradley, C., & Herman, J. (1985). *Intelligent computer-assisted instruction study* (JPL Contract #956881). Los Angeles: UCLA Center for the Study of Evaluation.
- Cohen, P., & Howe, A. (1988). How evaluation guides AI research. *AI Magazine*, 9 (4), pp. 35-43.
- Collins, A. (1987). *Reformulating testing to measure thinking and learning* (Report No. 6869). Cambridge, MA: BBN Systems and Technologies Corporation.
- Farr, M.J. (1986). *The long-term retention of knowledge and skills: A cognitive and instructional perspective* (IDA Memorandum Report M-205). Alexandria, VA: Institute for Defense Analyses.
- Feifer, R.G. (1989). *An intelligent tutoring system for graphic mapping strategies* (Technical Report UCLA-AI-89-04). Los Angeles: UCLA Computer Science Department.

- Glaser, R. (in press). Expertise and assessment. In M.C. Wittrock & E.L. Baker (Eds.), *Testing and cognition*. Englewood Cliffs, NJ: Prentice-Hall.
- Glennan, T.K., Jr. (1967). Issues in the choice of development policies. In T. Manschak, T.K. Glennan, Jr., & R. Summers (Eds.), *Strategies for research and development*. New York: Springer-Verlag.
- Lesgold, A., Bonar, J., & Ivill, J. (1987). *Toward intelligent systems for testing* (LRDC Technical Report ONR/LSP-1). Pittsburgh: University of Pittsburgh Learning Research and Development Center.
- Means, B., & Gott, S.P. (1988). Cognitive task analysis as a basis for tutor development: Articulating abstract knowledge representations. In J. Psotka, L.D. Massey & S.A. Mutter (Eds.), *Intelligent tutoring systems: Lessons learned*. Hillsdale, NJ: Lawrence Erlbaum.
- Read, W., Dyer, M.G., Baker, E.L., & Butler, F. (1989). *Natural language sourcebook*. Los Angeles: UCLA Center for Technology Assessment
- U.S. Department of Education Office of Educational Research and Improvement. (1988). *Creating responsible and responsive accountability systems: Report of the OERI State Accountability Study Group*. Washington, DC: Author.